FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Problemas de emparejamiento

Estela Vázquez-Monjardín Lorenzo

2018/2019

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Problemas de emparejamiento

Estela Vázquez-Monjardín Lorenzo

Setembro, 2019

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Coñecemento: Estadística e Investigación Operativa
Título: Problemas de emparejamiento
Breve descrición do contido
El objetivo de este trabajo es que el alumno se familiarice con una importante clase de problemas de programación lineal y entera: los problemas de emparejamiento. Se trata de una clase de problemas que engloban, como caso particular, al problema de asignación, visto en la asignatura de Programación Lineal y Entera. El alumno deberá familiarizarse bien con los distintos problemas de emparejamiento, y también con las similitudes y diferencias entre ellos que requerirán el uso de unas u otras técnicas de resolución.

Índice general

Resumen

A lo largo de esta memoria estudiaremos las distintas variantes del problema de emparejamiento y veremos algunas de sus aplicaciones prácticas. Incluimos una amplia introducción, en la que constan varios resultados sobre redes con flujo y algoritmos para resolver distintas versiones de problemas de optimización ya conocidos, que nos servirán de herramienta para el tema que nos ocupa. Este estudio incluirá resultados teóricos, como el teorema del camino aumentador, que nos permitirán presentar algoritmos especialmente diseñados para la resolución de los problemas de emparejamiento.

Abstract

In this paper we will study the different versions of the matching problem and we will show some of its practical applications. We include a wide introduction, in which there are several results about network flows and algorithms that solve various of the already known optimization problems, and will be a useful tool to the topic we are dealing with. We will include some theorical results, like the augmenting path theorem, that allow us to introduce some algorithms specially designed for matching problems.

Introducción

En nuestro día a día estamos en contacto continuo con distintas redes con flujo. Un problema de flujo en redes en un problema de optimización cuyo objetivo es mover elementos (unidades de flujo) de un punto a otro de la red y hacerlo de manera eficiente. Los orígenes de las redes con flujo se remontan a la segunda mitad del siglo XIX con el trabajo de Gustav Kirchhoff y otros pioneros de la ingeniería eléctrica, quienes analizaron por primera vez circuitos eléctricos. Estos trabajos establecieron muchas de las ideas clave de la teoría de las redes con flujo y consolidaron las redes o grafos como útiles objetos matemáticos para representar sistemas físicos. La Programación Lineal es la parte de la Investigación Operativa que se encarga del estudio de problemas de optimización, y su origen se remonta a la época de la Segunda Guerra Mundial cuando se utilizaba como herramienta para planificar operaciones logísticas y militares de manera eficiente. En las décadas de los 40 y los 50 las comunidades de investigadores desarrollaron la optimización como un campo independiente de investigación.

Los problemas de flujo en redes básicos (como el problema del flujo máximo o el problema de flujo en redes a coste mínimo) son problemas triviales desde el punto de vista de la matemática pura. Considerando un número finito de objetos para cada problema y una función objetivo definida sobre ellos, solamente debemos enumerar el conjunto de posibles soluciones y elegir aquella que sea mejor, es decir, que minimice (o maximice) el valor de la función objetivo. Sin embargo, lo que se busca son algoritmos que resuelvan en un tiempo *bueno* los problemas en la práctica. Durante las últimas décadas se ha conseguido un avance constante en esta búsqueda, encontrando nuevos métodos de resolución y mejoras de algunos métodos ya conocidos.

Este trabajo se centrará en el estudio de los problemas de optimización en redes conocidos como problemas de emparejamiento, y dedicaremos la mayor parte de la memoria a buscar algoritmos eficientes para resolverlos. Los problemas de emparejamiento aparecen en muchas situaciones diferentes, ya que a menudo intentamos encontrar la mejor manera de emparejar objetos o gente para alcanzar un objetivo. El problema clásico de empareja-

miento bipartito es un caso especial en el cual los objetos están separados en dos grupos, y deseamos emparejar los objetos de cada grupo con los del otro de alguna manera óptima. El problema de emparejamiento general modela situaciones en las que los objetos no están necesariamente divididos en dos grupos.

Con el fin de que el trabajo sea autocontenido, el Capítulo 1 es un repaso de conceptos vistos en la asignatura de Programación Lineal y Entera necesarios para el estudio de las distintas versiones de problemas de emparejamiento. También el Capítulo 2 será un capítulo introductorio, donde presentaremos conceptos no vistos en el Grado como los emparejamientos, caminos aumentadores o algoritmos de búsqueda.

Dedicaremos el Capítulo 4 a los problemas de emparejamiento bipartito. Consideraremos dos versiones: el problema de máxima cardinalidad, donde buscamos un emparejamiento que contenga el máximo número de arcos y el problema ponderado, donde cada arco tiene un coste asociado y buscamos un emparejamiento con el menor coste total.

Aunque en general los problemas de emparejamiento no pueden ser formulados como problemas de flujo en redes a coste mínimo, veremos que las versiones bipartitas son fáciles de resolver, pues la versión ponderada sí que es un caso particular de PFCM y la de máxima cardinalidad puede ser visto como un problema del flujo máximo tras una serie de modificaciones. Por esto dedicaremos el Capítulo 3 al estudio en profundidad del problema del flujo máximo.

Presentaremos un algoritmo para el problema del flujo máximo que resuelve el problema bipartito de máxima cardinalidad con n nodos y m arcos en un tiempo que crece a velocidad $\sqrt{n}m$, lo que normalmente se denota como $O(\sqrt{n}m)$ y cuyo significado preciso definiremos también en esta memoria. También mostraremos que, si $S(n, m, C)$ denota el tiempo necesario para resolver el problema del camino más corto en una red sin ciclos de longitud negativa con costes acotados por C, entonces una adaptación de algoritmos para el PFCM, resuelve el problema bipartito ponderado en un tiempo de $O(nS(n, m, nC))$.

En el Capítulo 5 estudiamos los problemas de emparejamiento no bipartito, que son más difíciles de resolver ya que no se pueden reducir a formas estándar de problemas de flujo en redes, por lo que necesitan algoritmos especialmente diseñados. Consideraremos la versión de máxima cardinalidad del problema de emparejamiento no bipartito, para el que describiremos un algoritmo que hace uso de los llamados caminos aumentadores para resolver este problema en un tiempo de $O(n^3)$.

Dado que la exposición de los algoritmos ocupa bastante extensión en el trabajo, nos hemos centrado menos en la discusión de aplicaciones pero, en cualquier caso, en el Anexo A presentamos algunos ejemplos de aplicaciones de los problemas de optimización estudiados.

Capítulo 1

Repaso

En este capítulo de repaso, con el objetivo de que el trabajo sea autocontenido, se recuerdan contenidos correspondientes a la asignatura de Programación Lineal y Entera del Grado en Matemáticas. Recordamos conceptos básicos sobre grafos, redes con flujo o el problema de flujo en redes a coste mínimo. Dado que el objetivo del trabajo es presentar varios algoritmos para resolver los distintos tipos de problemas de emparejamiento, haremos una breve introducción a la teoría de la complejidad computacional, que nos permitirá comparar la eficiencia de estos algoritmos. También comentaremos aspectos sobre la dualidad en problemas de optimización en redes, lo cual nos ayudará a entender mejor el funcionamiento de algunos de los algoritmos presentados.

Por ser todo este capítulo un repaso de cuestiones estudiadas en el Grado, la mayoría de la información se obtuvo de los apuntes González-Díaz (2018).

1.1. Introducción a la teoría de grafos y redes con flujo

Teoría de grafos

Un **grafo** G es un par (N, A) donde N representa el conjunto de **nodos** del grafo y A representa el conjunto de **arcos** entre los nodos. Denotaremos por $n = |N|$ el número total de nodos y $m = |A|$ el número total de arcos del grafo. Además, según cómo sean los arcos podemos distinguir entre:

- **Grafos no dirigidos**: los arcos son subconjuntos de dos elementos de N, es decir, como $\{i, j\}$ y $\{j, i\}$ son el mismo conjunto representan el mismo arco en la red.[1]

[1] A este arco no dirigido lo denotaremos por (i, j). Teniendo en cuenta que usaremos esta misma notación para los arcos dirigidos, es necesario ser conscientes del tipo de grafo en el que estamos trabajando en cada caso para evitar confusiones.

- **Grafos dirigidos** u **orientados**: en este caso $A \subseteq N \times N$, los arcos son pares ordenados de la forma (i, j), siendo i el nodo inicial y j el nodo final.

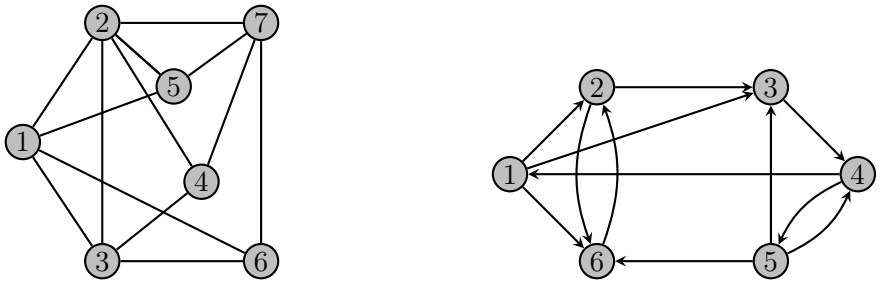


Figura 1.1: Ejemplos de grafos no dirigido y dirigido.

Dos nodos i y j son **adyacentes** si el arco (i, j) (o el (j, i)) está presente en el grafo G. En este caso, se dice que i y j son **incidentes** en el arco (i, j) y que el arco (i, j) es **incidente** en los nodos i y j. Se define la **lista de adyacencia** de un nodo i, que denotamos por A_i, al conjunto de aristas que se originan en i, es decir, $A_i = \{(i, j) \in A\}$. Nótese que en los grafos no dirigidos A_i coincide con el conjunto de arcos incidentes en i.

Un **grafo simple** es aquel que solo admite un arco entre cada par de nodos. Por ejemplo, en la figura anterior el primer grafo es simple, mientras que el segundo no lo es.

Un **subgrafo** de un grafo G es un grafo $G' = (N', A')$ que tiene todos sus arcos y nodos en G, es decir, $N' \subseteq N$ y $A' \subseteq A$.

Sea G un grafo no dirigido y sea $a_1 \hookrightarrow a_2 \hookrightarrow \ldots \hookrightarrow a_r$ una secuencia de arcos distintos. Si existen nodos $i_0 \hookrightarrow i_1 \hookrightarrow \ldots \hookrightarrow i_r$ tales que, para $l \in \{1, 2, \ldots, r\}$, $a_l = (i_{l-1}, i_l)$, decimos que la secuencia es una **cadena**. Podemos referirnos a la cadena tanto por la secuencia de arcos como por la secuencia de nodos que la forman. Además, existen varios tipos de cadenas:

- **Cadena cerrada**: es una cadena en la que $i_0 = i_r$.
- **Camino**: es una cadena en la que todos los nodos son distintos.
- **Circuito o ciclo**: es una cadena cerrada en la que no hay más nodos coincidentes que el primero y el último.

Un grafo es **conexo** si para cada par de vértices existe una cadena que los une.
Un grafo se dice que es un **árbol** si es conexo y no contiene ciclos.

Las definiciones que acabamos de presentar se pueden adaptar inmediatamente para grafos dirigidos.

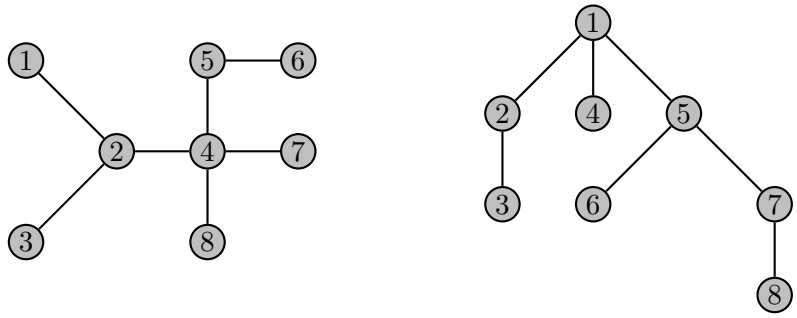


Figura 1.2: Dos ejemplos de árboles.

Dado un grafo dirigido $G = (N, A)$, se dice que G es un **grafo bipartito** si el conjunto de nodos se puede dividir en dos subconjuntos, $N = N_1 \cup N_2$, de manera que N_1 y N_2 son disjuntos y no vacíos. Además, si es dirigido, cada arco de A se origina en N_1 y termina en N_2, es decir, $A \subseteq N_1 \times N_2$. En la siguiente figura se muestran dos representaciones de un mismo grafo bipartito $G = (N_1 \cup N_2, A)$, donde $N_1 = \{1, 2, 3, 4\}$ y $N_2 = \{5, 6, 7, 8\}$.

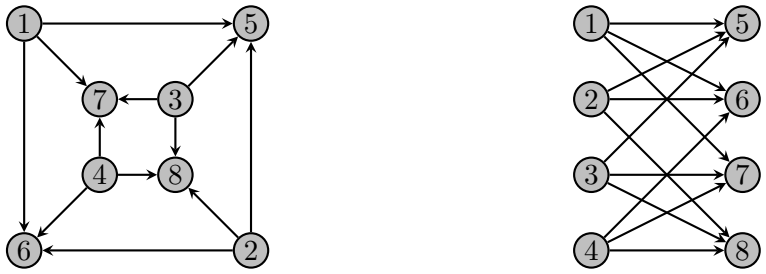


Figura 1.3: Dos representaciones del mismo grafo bipartito.

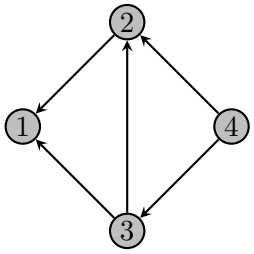
Para poder expresar problemas de optimización sobre grafos como problemas de programación lineal es conveniente que seamos capaces de resumir la información relativa a los grafos en forma de matrices. Sea $G = (N, A)$ un grafo dirigido, este puede ser representado por su **matriz de adyacencia** (nodo-nodo) $\overline{\mathbf{A}}_{n\times n}$, definida por:

$$\bar{a}_{ij} = \begin{cases} 1 & \text{si } (i,j) \text{ es un arco de } G \\ 0 & \text{en otro caso.} \end{cases}$$

Otra posible representación matricial del grafo G es a través de la **matriz de incidencia** (nodo-arco) $\overline{\mathbf{B}}_{n\times m}$, definida por:

$$\bar{b}_{ik} = \begin{cases} 1 & \text{si } i \text{ es el nodo inicial del arco } a_k \\ \text{-1} & \text{si } i \text{ es el nodo terminal del arco } a_k \\ 0 & \text{en otro caso.} \end{cases}$$

Por ejemplo, para el siguiente grafo se tienen las matrices de adyacencia e incidencia asociadas:



$$\overline{\mathbf{A}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \qquad \overline{\mathbf{B}} = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Redes con flujo

Una **red** es un grafo con uno o más números asociados a cada arco y/o nodo, que pueden representar precios, distancias u otros parámetros de interés. Las propiedades que definimos para los grafos se aplican también a redes, por ejemplo una red se dirá bipartita cuando el grafo subyacente lo sea.

Llamaremos **flujo** al envío de elementos de un lugar a otro de una red, nos referiremos a estos elementos que fluyen por la red como **unidades de flujo**. Las unidades de flujo pueden ser bienes, personas, información o casi cualquier cosa.

Asociados a cada arco tenemos varios parámetros importantes:
Flujo: f_{ij}, es el flujo que pasa por el arco (i, j).
Cota inferior: $l_{ij} \geq 0$, es la cantidad mínima de flujo que debe pasar por el arco (i, j).[2]
Cota superior o capacidad: $u_{ij} \geq l_{ij}$, es la cantidad máxima de flujo que puede pasar por el arco (i, j).
Coste: c_{ij}, es el coste de enviar una unidad de flujo por el arco (i, j). Si $c_{ij} < 0$, este parámetro representa beneficio por unidad de flujo.

Asociado a los nodos tenemos:
Suministro/demanda: b_i, es un número entero asociado al nodo $i \in N$ que representa suministro si $b_i > 0$, demanda si $b_i < 0$ o la existencia de un nodo de enlace si $b_i = 0$.

[2]Más adelante aclararemos por qué si este parámetro no aparece en la red se supone que es 0 y demostraremos que una red siempre es equivalente a otra con $l_{ij} = 0, \forall (i, j) \in A$.

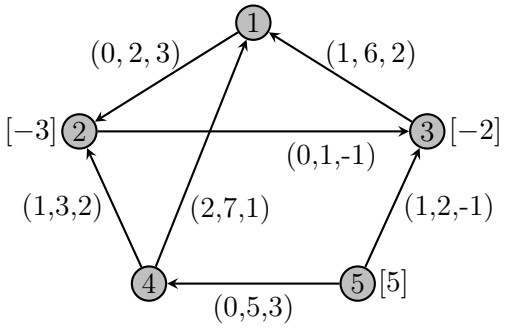


Figura 1.4: Ejemplo de red con flujo.

Como se observa en la figura anterior, asociada a cada arco se tiene una terna donde se muestran los parámetros de interés en el siguiente orden: (l_{ij}, u_{ij}, c_{ij}). Además, al lado de cada nodo aparece su suministro/demanda, que supondremos nulo en caso de no aparecer.

Entonces, una red con flujo R construida sobre el grafo $G = (N, A)$ vendrá caracterizada por una tupla $R = ((N, A), \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b})$, donde $\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}$ y $\boldsymbol{b}$ denotan los vectores de cotas inferiores, capacidades, costes y suministros/demandas respectivamente.

Durante todo el texto supondremos que $\sum_{i=1}^{n} b_i = 0$, lo cual quiere decir que no hay flujos externos en la red. En caso de presentársenos un problema en redes con flujos externos, podemos convertirlo fácilmente en un problema sin flujos externos sin más que añadir nodos ficticios. La explicación detallada de cómo hacer esta conversión se encuentra en González-Díaz (2018), en el apartado dedicado a las redes con flujo.

1.2. El problema de flujo en redes a coste mínimo (PFCM)

El problema de flujo en redes a coste mínimo (problema estudiado en profundidad en el Grado y al que nos referiremos como PFCM de ahora en adelante) es el problema de flujo en redes esencial, ya que muchos de los problemas de redes con flujo que aparecen en la vida real son casos particulares o generalizaciones del mismo.

Dada una red dirigida $R = ((N, A), \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b})$, un PFCM es un problema de optimización que consiste en determinar el flujo que ha de pasar por cada arco de la red, de modo que el coste asociado al flujo total sea mínimo y que además se satisfagan las demandas de ciertos nodos a partir de los suministros de otros, todo ello cumpliendo las restricciones de capacidad de los arcos.

Todo problema de flujo en redes a coste mínimo se puede formular como un problema de programación lineal, donde las variables de decisión son los flujos f_{ij} de cada arco, como sigue:

$$\begin{aligned}
\text{Minimizar} \quad & \sum_{(i,j)\in A} c_{ij} f_{ij} \\
\text{Sujeto a} \quad & \sum_{\{j:(i,j)\in A\}} f_{ij} - \sum_{\{j:(j,i)\in A\}} f_{ji} = b_i, \quad \text{para todo } i \in N \\
& l_{ij} \leq f_{ij} \leq u_{ij}, \quad \text{para todo } (i,j) \in A.
\end{aligned}$$

Obsérvese que tenemos:

- Un total de n restricciones de balance de flujo, una por cada nodo, que establecen que el flujo que sale de cada nodo menos el que entra debe ser igual a su suministro/demanda.
- Un total de $2m$ restricciones de capacidad, dos por cada arco, que exigen que cada arco alcance su cota inferior de flujo sin superar su capacidad.

El problema anterior se puede expresar matricialmente, usando la matriz de incidencia $\overline{\mathbf{B}}_{n\times m}$ del grafo, como sigue:

$$\begin{aligned}
\text{Minimizar} \quad & \boldsymbol{c}^{\mathsf{T}} \boldsymbol{f} \\
\text{Sujeto a} \quad & \overline{\mathbf{B}}_{n\times m} \boldsymbol{f} = \boldsymbol{b} \\
& \boldsymbol{l} \leq \boldsymbol{I}_{m\times m} \boldsymbol{f} \leq \boldsymbol{u},
\end{aligned}$$

donde $\boldsymbol{c}^{\mathsf{T}}$ (vector fila) es el vector traspuesto del vector de costes $\boldsymbol{c}$ e $\boldsymbol{I}_{m\times m}$ es la matriz identidad de tamaño m. Considerando esta expresión se ve claramente que estamos ante un caso particular de problema de programación lineal como los vistos en la primera parte de la asignatura de Programación Lineal y Entera.

En esa misma asignatura se estudia el método símplex para la resolución de problemas de programación lineal generales, que tiene un tiempo de ejecución en el peor caso exponencial, aunque solamente conlleva un tiempo mayor a $O(n^{3.5})$ en ejemplos patológicos.[3] Sin embargo, el método símplex no se aprovecha de la estructura adicional de la red subyacente a los PFCM, por lo que no es competitivo con otros algoritmos que han sido específicamente diseñados.

1.2.1. Casos particulares del PFCM

Algunos casos particulares del PFCM aparecen frecuentemente como subproblemas de otros problemas más complejos. A continuación, se presentan algunos de estos casos particulares:

[3] Explicaremos qué representa esta notación en la Sección 1.4 sobre complejidad computacional.

I. **El problema del camino más corto**

Este problema surge en el estudio de otros problemas de optimización y, debido a esto, muchos algoritmos para problemas más generales requieren la resolución de problemas del camino más corto en sus fases intermedias.

El problema del camino más corto consiste en encontrar un camino de coste mínimo entre un nodo fuente s y un nodo sumidero t, donde el coste de un camino es la suma de los costes de los arcos que lo forman.

Transformamos el problema del camino más corto en un PFCM general fijando todas las cotas inferiores a 0 y las superiores a 1; $b_s = 1, b_t = -1$ y $b_i = 0$ para el resto de nodos de la red. Así, la solución del problema de programación lineal enviará una unidad de flujo desde el nodo s al nodo t a lo largo del camino con menor coste.

Observación. El problema del camino más corto no se puede representar como un PFCM si la red contiene ciclos de longitud (coste) negativa. De hecho, este es un problema para el que no se conocen algoritmos polinomiales.[4]

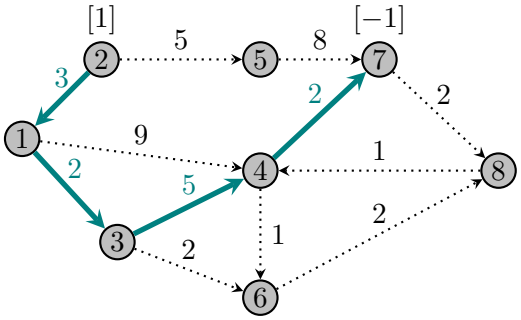


Figura 1.5: Uno de los caminos de mínima longitud entre los nodos 2 y 7, donde en cada arista a_{ij} se explicita el coste por unidad de flujo c_{ij}.

Este problema tiene muchas aplicaciones distintas, algunas más intuitivas como encontrar las rutas más rápidas de conducción entre distintos lugares o las rutas más baratas para enviar cargas. Otra aplicación menos intuitiva es la resolución del cubo de Rubik en el menor número de movimientos posibles. En este caso los nodos representarían los distintos estados del cubo, mientras que los arcos representarían movimientos o giros, así, los algoritmos para el problema del camino más corto encontrarían la manera de resolver el cubo utilizando el menor número de movimientos a partir de una posición dada.

[4]Explicaremos en la Sección 1.4 qué significa que un algoritmo sea polinomial.

II. **El problema del flujo máximo**

El problema del camino más corto modela situaciones donde no nos preocupan las restricciones de capacidad, pues por un arco se puede enviar o no enviar flujo, pero está sometido a costes. El problema del flujo máximo es complementario al problema del camino más corto, pues en él nos despreocupamos de los costes y sin embargo sí que existen restricciones de capacidad.

El problema del flujo máximo busca una solución que envíe la máxima cantidad de flujo desde un nodo fuente s hasta un nodo sumidero t, respetando las cotas de capacidad, $l_{ij} \leq f_{ij} \leq u_{ij}$, en cada arco de la red. Este problema también aparece a menudo como subproblema de otros más complicados, por lo que se estudiará más a fondo en el Capítulo 3.

Transformamos el problema del flujo máximo en un PFCM general fijando $b_i = 0$ para todo $i \in N$ y $c_{ij} = 0$ para todo $(i, j) \in A$. Además, debemos introducir en la red un "arco de vuelta" del sumidero a la fuente, con coste negativo $c_{ts} = -1$ y capacidad infinita $u_{ts} = \infty$, de modo que será beneficioso enviar flujo de s a t. Así, la solución del problema maximiza el flujo que pasa por el arco (t, s), o lo que es lo mismo, maximiza la cantidad de flujo que va desde s hasta t por arcos de la red original.

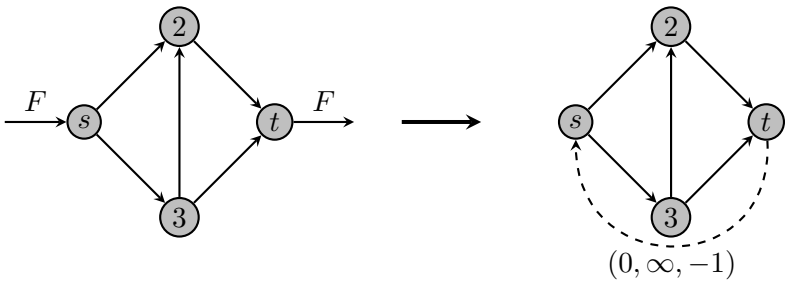


Figura 1.6: Transformando un PFM en un PFCM.

Una de las aplicaciones más intuitivas de este problema es la de transportar bienes de un lugar a otro a través de una red de tuberías, carreteras, etc. Si una empresa de gas quiere calcular la máxima cantidad de gas que puede transportar de una ciudad (fuente) a otra (sumidero) a través de su red de gasoductos, tendrá que resolver el problema del flujo máximo donde las restricciones de capacidad de cada arco vienen dadas por el diámetro del gasoducto correspondiente. Para modelar este problema también habría que tener en cuenta otras restricciones físicas, por ejemplo las asociadas a la presión del gas en los distintos nodos de la red.

III. **El problema del transporte**

Partimos de una red bipartita R, donde N_1 y N_2 pueden tener distinto número de elementos. En este problema cada nodo de N_1 es un nodo de suministro ($b_i > 0$) y cada nodo de N_2 es un nodo de demanda ($b_j < 0$). El coste de cada arco $(i, j) \in A$ puede representar, por ejemplo, las distancia entre los nodos i y j.

El problema del transporte busca una solución que envíe unidades de flujo desde los nodos suministro en N_1 para satisfacer las demandas de N_2 a un menor coste. Para que el problema tenga solución, claramente las demandas no podrán exceder a los suministros, es decir, debe cumplirse $\sum_{i \in N_1} b_i \geq \sum_{j \in N_2} b_j$.

En una de las aplicaciones más comunes del problema del transporte los nodos de suministro representan fábricas de bienes materiales o plantas energéticas, mientras que los nodos de demanda son los centros de almacenamiento de dichos bienes.

IV. **El problema de asignación**

Es un caso particular del problema del transporte y también del problema de emparejamiento, por lo que lo estudiaremos más ampliamente en la Sección 4.2.

En este caso se parte de una red bipartita donde $|N_1| = |N_2|$ y los arcos, $A \subseteq N_1 \times N_2$, representan las posibles asignaciones, cada una con coste c_{ij}. Se busca emparejar, con el menor coste posible, cada elemento de N_1 con un elemento de N_2.

Transformamos el problema de asignación en un PFCM general fijando: $b_i = 1$ para los nodos de N_1, $b_i = -1$ para los nodos de N_2, $l_{ij} = 0$ y $u_{ij} = 1$ para todos los arcos de A. Así, la solución del problema establecerá una correspondencia entre los nodos de N_1 y los de N_2 al menor coste posible.

1.2.2. PFCM con variables enteras: Unimodularidad

En muchas aplicaciones de los problemas de flujo en redes en la vida real los flujos representan objetos indivisibles. En este caso, al modelar el problema como un PFCM, los flujos deben tomar valores enteros y tenemos un problema de programación lineal entera. Estos problemas no son fáciles en general, sin embargo la estructura adicional de los PFCM nos permite sortear este inconveniente gracias a la propiedad de unimodularidad.

Una matriz cuadrada $\boldsymbol{A} \in \mathbb{Z}^{p \times p}$ se dice **unimodular** si su determinante es 1 o -1.

Una matriz $\boldsymbol{A} \in \mathbb{Z}^{p \times q}$ se dice **totalmente unimodular** si cualquier submatriz cuadrada es singular o unimodular. Esta condición ha de cumplirse también para las submatrices de tamaño 1×1, por lo que cada elemento de una matriz totalmente unimodular ha de ser 0, 1 o -1, es decir, $\boldsymbol{A} \in \{-1, 0, 1\}^{p \times q}$.

A continuación se muestran varios resultados que ilustran cómo se relaciona la propiedad de unimodularidad de una matriz con las soluciones de un PFCM. Las demostraciones están escritas en detalle en González-Díaz (2018).

Proposición 1.1. *Dados una matriz totalmente unimodular $\boldsymbol{A} \in \mathbb{Z}^{p\times q}$ y un vector $\boldsymbol{b} \in \mathbb{Z}^p$, tenemos que cualquier solución básica factible definida por las restricciones $\boldsymbol{Ax} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$, tiene todas sus componentes enteras.*

Corolario 1.2. *Dado un problema de programación lineal en forma estándar ($\boldsymbol{Ax} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$), si la matriz de restricciones es totalmente unimodular y el vector de lados derechos es entero, entonces todo punto extremo de la región factible tiene todas sus componentes enteras. En particular, si el problema tiene óptimos finitos, al menos uno será entero.*

La estructura de los PFCM nos permite usar estos resultados para concluir lo siguiente.

Proposición 1.3. *La matriz de incidencia $\overline{\boldsymbol{B}}$ de un grafo dirigido es totalmente unimodular.*

Proposición 1.4. *Si todas las capacidades de un PFCM toman valores enteros, entonces habrá al menos un óptimo entero.*

Por último, nótese que la hipótesis de que las capacidades sean enteras no es restrictiva. Podemos convertir cualquier PFCM en otro equivalente con capacidades enteras sin más que redondear cada cota superior u_{ij} al entero más próximo no mayor que u_{ij} y redondear cada cota inferior l_{ij} al entero más próximo no menor que l_{ij}. Además estos cambios no afectarán al óptimo del problema, que sabemos que existe por el resultado anterior.

1.3. Dualidad en problemas de optimización en redes

La dualidad es una herramienta fundamental para el estudio de problemas de optimización y fuente de inspiración para el desarrollo de algoritmos eficientes. A todo problema de programación lineal le podemos asociar un nuevo problema de programación lineal, conocido como el **problema dual**, con importantes propiedades. Si llamamos primal al problema original, podemos usar las propiedades del dual para ayudarnos a resolver el problema primal.

Las demostraciones de los resultados que presentamos en esta sección se pueden consultar en González-Díaz (2018).

Formulaciones en forma canónica de un problema (P) y su dual (D):

Problema primal (P)		**Problema dual** (D)	
Minimizar	$\boldsymbol{c}^\top \boldsymbol{x}$	Maximizar	$\boldsymbol{b}^\top \boldsymbol{\pi}$ $(\Leftrightarrow \boldsymbol{\pi}^\top \boldsymbol{b})$
Sujeto a	$\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$	Sujeto a	$\boldsymbol{A}^\top \boldsymbol{\pi} \leq \boldsymbol{c}$ $(\Leftrightarrow \boldsymbol{\pi}^\top \boldsymbol{A} \leq \boldsymbol{c}^\top)$
	$\boldsymbol{x} \geq \boldsymbol{0}$,		$\boldsymbol{\pi} \geq \boldsymbol{0}$.

De donde se puede deducir que:

- Si el primal es un problema de minimización, entonces el dual lo es de maximización y viceversa.
- Variables y restricciones se intercambian los papeles. Cada variable del primal se corresponde con una restricción del dual y cada restricción del primal se corresponde con una variable del dual.
- El vector $\boldsymbol{b}$ de lados derechos del primal es el vector de costes del dual y el vector de costes del primal $\boldsymbol{c}$ es el vector de lados derechos en el dual.

Dualidad débil y dualidad fuerte

A continuación presentamos un par de resultados que relacionan las soluciones del problema primal y las del dual.

Teorema 1.5. *(Dualidad débil). Dado un par de problemas duales, (P) y (D), la función objetivo asociada a cualquier solución factible del problema de minimización es mayor o igual que la función objetivo asociada a cualquier solución factible del problema de maximización.*

Demostración. Suponemos, sin pérdida de generalidad, que tenemos el problema primal expresado en forma canónica y que $\boldsymbol{x}$ y $\boldsymbol{\pi}$ son soluciones factibles del primal y del dual, respectivamente. Entonces, $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{\pi}^\top \boldsymbol{A} \leq \boldsymbol{c}^\top$ y $\boldsymbol{\pi} \geq \boldsymbol{0}$. Aplicando estas relaciones a la función objetivo del primal obtenemos que

$$\boldsymbol{c}^\top \boldsymbol{x} \geq (\boldsymbol{\pi}^\top \boldsymbol{A})\boldsymbol{x} = \boldsymbol{\pi}^\top (\boldsymbol{A}\boldsymbol{x}) \geq \boldsymbol{\pi}^\top \boldsymbol{b}.$$

□

Este resultado implica que las soluciones factibles del primal siempre nos darán cotas superiores para las del dual. Análogamente, las soluciones factibles del dual siempre nos darán cotas inferiores para las del primal. Esta relación se cumple para cualquier par de problemas de optimización duales entre sí, sean o no lineales, y es de gran relevancia para el desarrollo de algoritmos.

Corolario 1.6. *Si $\boldsymbol{x}$ y $\boldsymbol{\pi}$ son soluciones factibles del primal y del dual tales que $\boldsymbol{c}^\top \boldsymbol{x} = \boldsymbol{\pi}^\top \boldsymbol{b}$, entonces $\boldsymbol{x}$ y $\boldsymbol{\pi}$ son soluciones óptimas del primal y del dual respectivamente.*

El siguiente teorema, a diferencia de la propiedad de dualidad débil, no siempre se cumple para problemas de optimización no lineal.

Teorema 1.7. *(Dualidad fuerte). Dado un par de problemas duales, (P) y (D), si uno de ellos tiene una solución óptima, entonces también el otro tiene solución óptima y los valores óptimos de la función objetivo coinciden. En particular, si $\bar{\boldsymbol{x}}$ y $\bar{\boldsymbol{\pi}}$ son soluciones óptimas de (P) y (D) respectivamente, tenemos que:*

$$\boldsymbol{c}^\top \bar{\boldsymbol{x}} = \bar{\boldsymbol{\pi}}^\top \boldsymbol{b}.$$

Holguras complementarias

Cuanto más severa es una restricción en uno de los problemas, más libertad tiene la variable asociada del otro y viceversa. El teorema que presentamos a continuación caracteriza cómo se resuelve esta tensión en un par primal-dual óptimo y es la base sobre la que se sustentan algunos de los algoritmos para problemas de optimización en redes.

Teorema 1.8. *(Teorema de las holguras complementarias). Dado un par de problemas duales, (P) y (D), y un par de soluciones factibles $\boldsymbol{x}$ y $\boldsymbol{\pi}$. Entonces, $\boldsymbol{x}$ y $\boldsymbol{\pi}$ forman un par de soluciones óptimas si y solo si*

$$\pi_i(\boldsymbol{A}_i^\top \boldsymbol{x} - b_i) = 0 \quad \textit{para todo } i \in \{1, \ldots, n\},$$

$$(c_j - \boldsymbol{\pi}^\top \boldsymbol{A}_j)x_j = 0 \quad \textit{para todo } j \in \{1, \ldots, m\},$$

donde, dada una matriz $\boldsymbol{A}$ denotamos su fila i-ésima con el vector $\boldsymbol{A}_i$ y su columna j-ésima con el vector $\boldsymbol{A}_j$.

Este resultado implica que, para un par de óptimos $\boldsymbol{x}$ y $\boldsymbol{\pi}$, si una restricción del dual no se satura, entonces la correspondiente variable del primal será 0. Mientras que si una variable no negativa del primal es estrictamente positiva, entonces la correspondiente restricción en el dual estará saturada.

Las variables del problema dual tienen la siguiente interpretación en el primal: sus valores indican cuánto estaría dispuesta a pagar por cada unidad en la que pudiese relajar el lado derecho de la restricción asociada, motivo por el cual es habitual llamarlos *precios sombra* y denotarlos por la letra π.

Dual de un PFCM sin cotas

Formulación del dual de un PFCM sin cotas:

Problema primal		**Problema dual**	
Minimizar	$\boldsymbol{c}^{\mathsf{T}}\boldsymbol{f}$	Maximizar	$\boldsymbol{\pi}^{\mathsf{T}}\boldsymbol{b}$
Sujeto a	$\overline{\mathbf{B}}_{n\times m}\boldsymbol{f} = \boldsymbol{b}$	Sujeto a	$\overline{\mathbf{B}}^{\mathsf{T}}_{n\times m}\boldsymbol{\pi} \leq \boldsymbol{c}$
	$\boldsymbol{f} \geq \mathbf{0}$.		$\boldsymbol{\pi} \in \mathbb{R}^n$.

Nótese que en el problema dual tenemos tantas restricciones como aristas y tantas variables como nodos. Además, dado que cada fila de $\overline{\mathbf{B}}^{\mathsf{T}}_{n\times m}$ tiene únicamente dos entradas distintas de cero, cada restricción del dual involucra únicamente a dos nodos.

El problema primal consiste en lo siguiente:

- Decidir el flujo que circula por cada arista $(i,j) \in A$, f_{ij}.
- Minimizando el coste total $\sum_{(i,j)\in A} c_{ij} f_{ij}$.
- Respetando las restricciones de conservación de flujo, una por cada nodo $i \in N$.

El dual podemos pensarlo como un problema de compra-venta de nodos, donde $b_i > 0$ indica de cuántas unidades del nodo i disponemos para vender y $b_j < 0$ indica las unidades que queremos comprar del nodo j. Por tanto, el problema dual consiste en lo siguiente:

- Decidir el precio que se le asigna a cada nodo $i \in N, \pi_i$.
- Maximizando los beneficios de las operaciones de compra-venta $\sum_{i\in N} \pi_i b_i$.
- Si pensamos c_{ij} como el coste asociado a un trueque disponible del nodo i por el nodo j, los precios π_i han de ser competitivos con dichos costes, es decir:

$$\pi_i - \pi_j \leq c_{ij}, \text{ para todo } (i,j) \in A.$$

1.4. Complejidad computacional

A lo largo de este trabajo presentamos varios algoritmos para resolver distintos tipos de problemas, algoritmos cuya efectividad puede depender del tamaño de la instancia a la que se aplique.[5] La rama de las matemáticas que estudia esta efectividad es la llamada teoría de la complejidad computacional, que en la década de los 70 empezó a estudiarse formalmente y ha dado lugar a uno de los campos más activos actualmente en las matemáticas.

[5] Llamaremos *instancia* a cada caso concreto de un problema, mientras que la palabra *problema* la reservaremos para referirnos a toda una clase de problemas, como el problema del camino más corto o el problema de emparejamiento bipartito.

La teoría de la complejidad estudia cómo crece el coste computacional de resolver un determinado problema en relación a lo que crece el tamaño de dicho problema. Cuando hablamos de coste computacional de un problema nos referimos a la memoria y el tiempo requerido por un ordenador para resolverlo. Es importante notar que no es lo mismo la complejidad de un algoritmo y la complejidad de un problema, pues habrá distintos algoritmos que resuelvan el mismo problema con distinta efectividad. La complejidad computacional de un problema será la complejidad del mejor algoritmo conocido que lo resuelva.

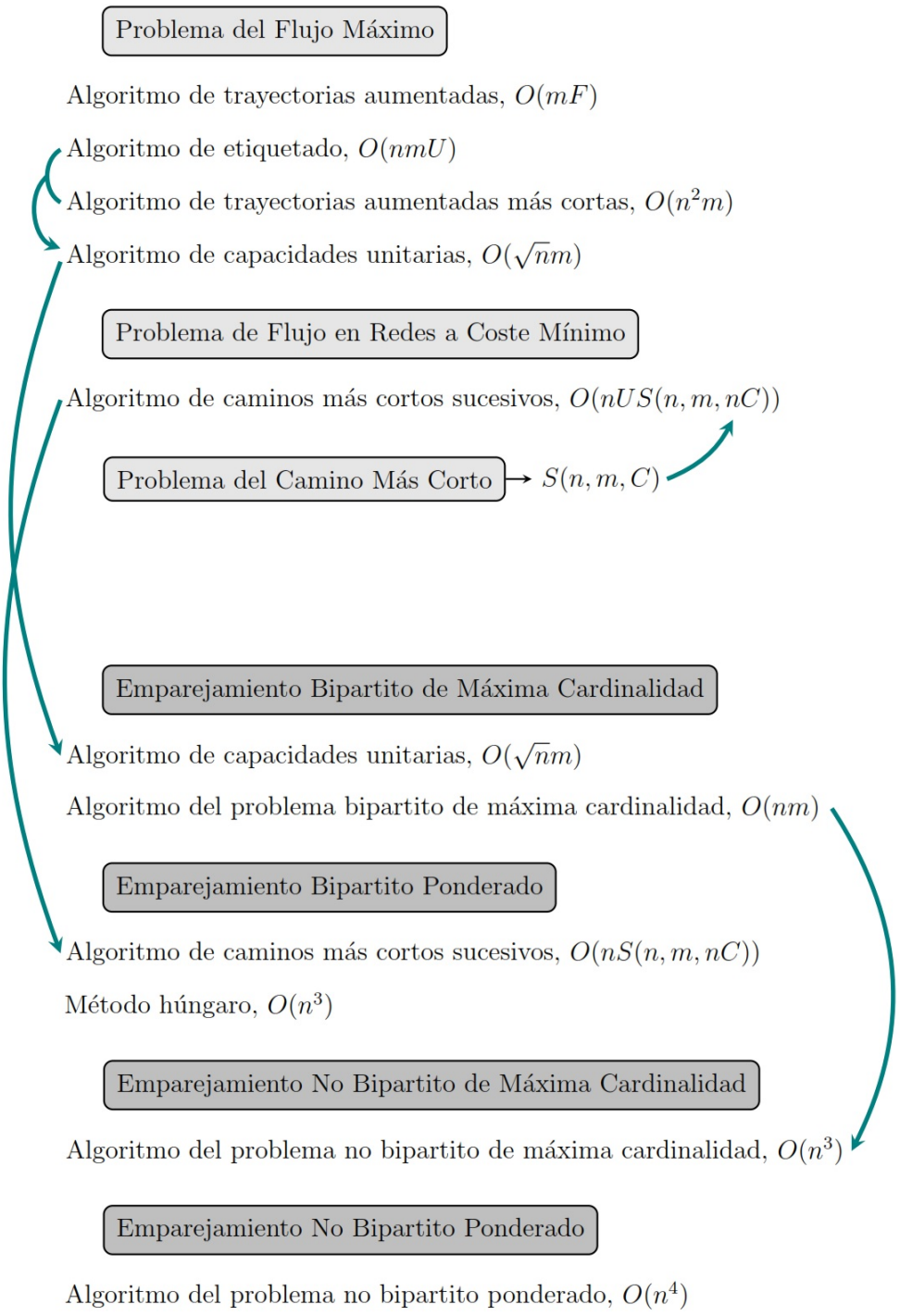
Para estudiar la eficiencia de un algoritmo debemos acotar la cantidad de operaciones que necesita en función del tamaño de los datos necesarios para representar el problema. Diremos que un algoritmo tiene **velocidad** $O(f(n))$ para resolver un problema si existen constantes k y n_0 tales que el número de operaciones efectuadas por el algoritmo para resolver cualquier instancia de tamaño $n \geq n_0$ es, a lo sumo, $kf(n)$. Un **algoritmo de tiempo polinomial** es aquel cuya velocidad es $O(p(n))$ para alguna función polinómica p. En la práctica nos interesa encontrar algoritmos cuyo tiempo de computación sea, como mucho, polinomial, pues los algoritmos no polinomiales pueden ser muy poco útiles para estudiar instancias grandes. De más rápida a más lenta, la velocidad de un algoritmo puede ser: constante, logarítmica, polinómica, pseudo-polinómica, exponencial o peor. En el último caso, con un pequeño crecimiento de los datos el coste computacional se hace inasumible. Los algoritmos que tienen un tiempo polinomial o menor se llaman **eficientes**. Además, el orden de convergencia de un algoritmo lo marca el término que contenga la función que crece más rápido con n. Por ejemplo, un algoritmo que necesita $\log(n)+100n+n^2+0.001n^3$ operaciones para resolver una instancia de tamaño n tiene velocidad $O(n^3)$.

Existen varios enfoques para medir el rendimiento de un algoritmo. Nosotros trabajaremos con el **análisis del peor caso**, que consiste en encontrar cotas superiores para el número de operaciones que va a necesitar el algoritmo para resolver cualquier instancia dentro del problema en estudio. El inconveniente de este enfoque es que puede catalogar como malo un algoritmo que solo funciona mal en instancias patológicas.

Veamos ahora un ejemplo que ilustre a qué nos referimos cuando hablamos de **algoritmos pseudo-polinomiales**. Si C denota el mayor coste de la red, un algoritmo cuya velocidad depende linealmente de C no puede considerarse polinomial ya que, si definimos $b = \log_2(C)$ como el número de bits necesarios para almacenar C, la complejidad asociada a un algoritmo de la forma $O(n^2C)$ realmente sería $O(n^2 2^b)$, que es exponencial. En cualquier caso, si sabemos que C no crece muy rápido a medida que n y m aumentan, los algoritmos pseudo-polinomiales pueden ser muy efectivos.

A continuación se muestra una figura con todos los algoritmos presentados a lo largo del trabajo y las relaciones entre ellos, donde las flechas indican que el primero de los

algoritmos sirve de herramienta para el desarrollo del siguiente. También se incluye al lado de cada algoritmo su velocidad de ejecución, donde F, U y C representan parámetros cuyo significado aclararemos en cada sección correspondiente.



Capítulo 2

Preliminares

En este capítulo introduciremos varias maneras de simplificar redes con flujo; algunas definiciones relativas a teoría de grafos que no se estudian en el Grado; algoritmos de búsqueda, que sirven como herramienta para el desarrollo de varios de los algoritmos que presentaremos a lo largo del trabajo y un algoritmo para el PFCM en el cual nos apoyaremos para resolver cierto tipo de problemas de emparejamiento.

2.1. Transformaciones de redes

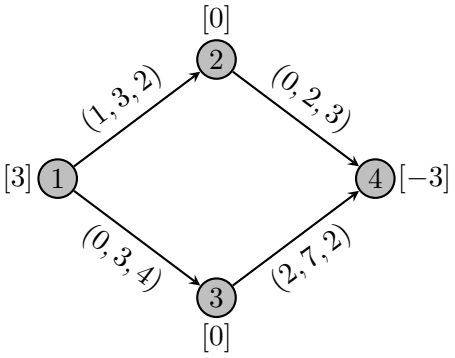
En muchos casos realizamos transformaciones sobre las redes con flujo para facilitar la resolución de problemas sobre ellas. Esta sección está basada en el Capítulo 2 de Ahuja et al. (1993).

2.1.1. Eliminando las cotas inferiores no nulas

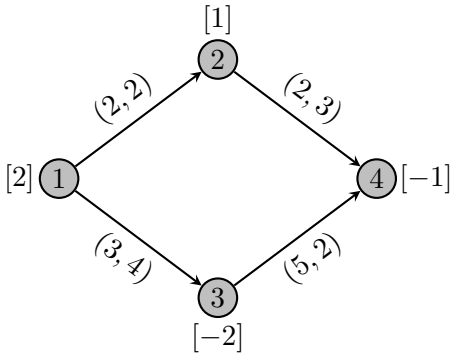
En algunos de los problemas de flujo en redes que trataremos solo aparecen dos parámetros asociados a cada arco. Estos parámetros son (u_{ij}, c_{ij}) ya que presuponemos, sin pérdida de generalidad, que $l_{ij} = 0$ para todos los arcos de la red. Si se nos presenta una red $R = ((N, A), \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b})$ con cotas inferiores positivas podemos convertirla fácilmente en otra red equivalente $R^* = ((N, A), \mathbf{0}, \boldsymbol{u}^*, \boldsymbol{c}^*, \boldsymbol{b}^*)$ con cotas inferiores nulas. Si la arista (i, j) tiene cota inferior positiva $l_{ij} > 0$ modificamos los parámetros de R del siguiente modo:

- $l^*_{ij} = 0$.
- $u^*_{ij} = u_{ij} - l_{ij}$.
- $b^*_i = b_i - l_{ij}$.
- $b^*_j = b_j + l_{ij}$.

Después de repetir este procedimiento tantas veces como aristas haya en R con $l_{ij} > 0$, obtenemos la red equivalente R^*. Por tanto, si partimos de un problema de optimización sobre R, esta transformación da lugar a un problema de optimización equivalente sobre R^*. Al resolverlo obtendremos un flujo $\boldsymbol{f}^*$, que se corresponde con un flujo en la red R obtenido sumando a cada componente de $\boldsymbol{f}^*$ la cota inferior l_{ij} correspondiente.



(a) Red R con $\boldsymbol{l} = (1, 0, 0, 2)$.



(b) Red R^* con $\boldsymbol{l}^* = \boldsymbol{0}$.

Modificando los parámetros de los arcos $(1, 2)$ y $(3, 4)$, y por tanto los suministros/demandas de los cuatro nodos, conseguimos una red R^* con cotas inferiores nulas. Si queremos abastecer los nodos con demanda a partir de los suministros de los nodos con $b_i > 0$, sobre la nueva red obtenemos un flujo $\boldsymbol{f}^* = (f_{12}, f_{13}, f_{24}, f_{34}) = (0, 2, 1, 0)$. Podemos obtener el flujo correspondiente en la red original sin más que hacer $f_{ij} = f^*_{ij} + l_{ij}$, de modo que en este ejemplo obtenemos $\boldsymbol{f} = (1, 2, 1, 2)$ con coste asociado 17.

2.1.2. Trabajando con costes reducidos

Sea $R = ((N, A), \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b})$ la red sobre la cual se plantea un PFCM. Recordemos que en el problema dual de un PFCM (Sección 1.3) las variables, $\pi_i, i \in N$, deben cumplir $\pi_i - \pi_j \leq c_{ij}$, es decir, $c_{ij} - \pi_i + \pi_j \geq 0,\ \forall (i, j) \in A$. Definimos:

- **Potenciales de los nodos**: π_i, es un número entero asociado a cada nodo $i \in N$ que representa cuánto estaría dispuesta a pagar por cada nodo i en el problema dual. El vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ es el vector de potenciales.
- **Costes reducidos**: $c^{\boldsymbol{\pi}}_{ij}$, es un número asociado a cada arco $(i, j) \in A$ que viene dado por $c^{\boldsymbol{\pi}}_{ij} = c_{ij} - \pi_i + \pi_j \geq 0$. Representa la holgura que tenemos en la restricción del dual asociada a la arista $(i, j) \in A$ con respecto a los costes del primal.

A continuación veremos un resultado que afirma que, cuando se trata de problemas de flujo en redes a coste mínimo, podemos trabajar en una red cuyos costes son los costes reducidos de otros dados y obtendremos la solución al problema inicial. Es decir, si queremos resolver un problema de optimización cuya función objetivo es $z(\mathbf{0}) = \sum_{(i,j)\in A} c_{ij} f_{ij}$, podemos buscar las soluciones del problema que tiene como función objetivo $z(\boldsymbol{\pi}) = \sum_{(i,j)\in A} c_{ij}^{\boldsymbol{\pi}} f_{ij}$. Formalicemos este hecho:

Propiedad 2.1. *Dos problemas de flujo en redes a coste mínimo sobre una misma red, con costes c_{ij} y $c_{ij}^{\boldsymbol{\pi}}$ respectivamente, tienen las mismas soluciones óptimas. Es más, siendo $z(\mathbf{0})$ la función objetivo del problema original y $z(\boldsymbol{\pi})$ la función objetivo del problema correspondiente a los costes reducidos, se cumple $z(\boldsymbol{\pi}) = z(\mathbf{0}) - \boldsymbol{\pi}\boldsymbol{b}$.*

Demostración. Para entender la relación entre las funciones objetivo $z(\mathbf{0})$ y $z(\boldsymbol{\pi})$, supongamos que partimos de un vector de potenciales $\boldsymbol{\pi} = \mathbf{0}$ e incrementamos el potencial del nodo k hasta π_k. Por definición de $c_{ij}^{\boldsymbol{\pi}}$, este cambio disminuye el coste reducido de cada unidad de flujo que sale del nodo k en π_k unidades y aumenta el coste reducido de cada unidad de flujo que entra en k en π_k unidades.

Por tanto el coste total (el valor de la función objetivo $z(\mathbf{0})$) disminuye en total π_k veces el flujo saliente menos el flujo entrante al nodo k. Así, para todo flujo cumpliendo las restricciones de balance de flujo, el valor de $z(\mathbf{0})$ disminuye en $\pi_k b_k$ unidades.

Repitiendo este proceso para cada nodo $i \in N$, se obtiene que

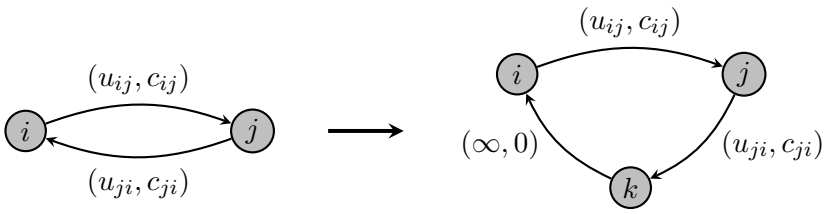
$$z(\boldsymbol{\pi}) = z(\mathbf{0}) - \sum_{i\in N} \pi_i b_i = z(\mathbf{0}) - \boldsymbol{\pi}\boldsymbol{b}.$$

Para un vector de potenciales dado $\boldsymbol{\pi}\boldsymbol{b}$ es constante, por tanto un flujo que minimice $z(\boldsymbol{\pi})$ también minimiza $z(\mathbf{0})$. Por esto, ambos PFCMs tendrán las mismas soluciones óptimas. □

2.1.3. Convirtiendo una red general en una red simple

En la siguiente sección nos referimos a la construcción de redes residuales a partir de una red R y un flujo $\boldsymbol{f}^0$, la cual puede presentar dificultades si no imponemos que la red original sea simple.[1] Veamos que, dado un problema de optimización en redes sobre una red R (que puede tener más de un arco entre cada par de nodos) es posible obtener otro problema equivalente sobre una red simple. Para convertir una red con costes general en una red simple equivalente, para cada par de nodos $i, j \in N$ con más de un arco entre ellos se añade un nuevo nodo k y se reestablecen los costes y capacidades del siguiente modo:

[1] En ese caso, en $R^{\boldsymbol{f}^0}$ existirán 2 arcos desde el nodo i hasta el j, cada uno con distintos costes y capacidades, y otros 2 arcos desde el nodo j hasta el i, también con distintos costes y capacidades.



En particular cuando tratamos el problema del flujo máximo estas dificultades no aparecen, pues los costes de todos los arcos son nulos y por tanto podemos combinar los distintos arcos entre i y j sin más que sumar sus capacidades.

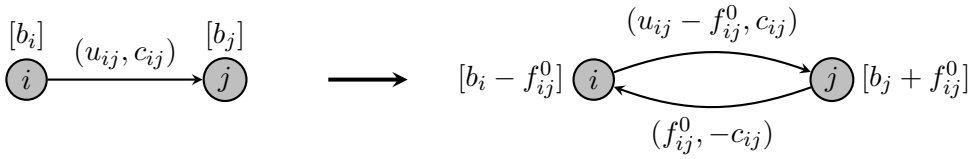
2.1.4. Trabajando con redes residuales

Un concepto de suma importancia en el diseño de algoritmos es el de las redes residuales. En cada paso se construye una red auxiliar, llamada **red residual**, que funciona como una red de *flujos restantes*. Veremos una propiedad que establece que las formulaciones de un problema de optimización sobre la red original y sobre la red residual son equivalentes, ya que se establece una correspondencia entre las soluciones factibles de ambos problemas. Denotaremos por $R^{\boldsymbol{f}^0}$ a la red residual correspondiente a la red R y el flujo $\boldsymbol{f}^0$. A continuación damos una idea intuitiva de cómo se construye.

Si por el arco (i, j) se envían f^0_{ij} unidades de flujo, podemos enviar por él $u_{ij} - f^0_{ij}$ unidades de flujo adicionales. También podemos enviar hasta f^0_{ij} unidades de flujo del nodo j al nodo i a través del arco (i, j), lo cual equivale a eliminar el flujo existente en ese arco. Mientras que enviar una unidad de flujo a través de (i, j) aumenta el coste total en c_{ij} unidades, enviar flujo de j a i por el mismo arco disminuye el coste en c_{ij} unidades.

Usando estas ideas, partiendo de una red simple $R = ((N, A), \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b}))$ y un flujo $\boldsymbol{f}^0$, se construye la red residual $R^{\boldsymbol{f}^0} = ((N', A'), \boldsymbol{u}', \boldsymbol{c}', \boldsymbol{b}')$ como sigue:

- R y $R^{\boldsymbol{f}^0}$ tienen el mismo conjunto de nodos, $N = N'$.
- Si el arco (i, j) está en R y $f^0_{ij} < u_{ij}$, entonces incluimos el arco (i, j) en $R^{\boldsymbol{f}^0}$ con coste $c'_{ij} = c_{ij}$ y capacidad residual: $u'_{ij} = u_{ij} - f^0_{ij}$.
- Si el arco (i, j) está en R y $f^0_{ij} > 0$, entonces incluimos el arco (j, i) en $R^{\boldsymbol{f}^0}$ con coste $c'_{ji} = -c_{ij}$ y capacidad residual: $u'_{ji} = f^0_{ij}$. Además actualizamos los suministros/demandas del siguiente modo: $b'_i = b_i - f^0_{ij}$ y $b'_j = b_j + f^0_{ij}$.



Dados un flujo $\boldsymbol{f}^0$ a partir del cual se construye la red residual y un flujo $\boldsymbol{f}$ sobre R, definimos el flujo $\boldsymbol{f'} \geq \boldsymbol{0}$ sobre $R^{\boldsymbol{f}^0}$ como el flujo que cumple:

$$f'_{ij} - f'_{ji} = f_{ij} - f^0_{ij} \tag{2.1}$$

y

$$f'_{ij} f'_{ji} = 0. \tag{2.2}$$

El siguiente resultado nos garantiza que todo flujo factible en R se corresponde con un flujo factible en $R^{\boldsymbol{f}^0}$ y además establece una relación entre los costes en ambas redes.

Propiedad 2.2. *Un flujo $\boldsymbol{f}$ es un flujo factible en R si, y solo si, el flujo correspondiente $\boldsymbol{f'}$ es factible en la red residual $R^{\boldsymbol{f}^0}$. Además, $\boldsymbol{cf} = \boldsymbol{c'f'} + \boldsymbol{cf}^0$.*

Demostración. La condición (2.2) implica que f'_{ij} y f'_{ji} no pueden ser positivos simultáneamente. Si $f_{ij} \geq f^0_{ij}$ entonces fijamos $f'_{ij} = f_{ij} - f^0_{ij}$ y $f'_{ji} = 0$. En este caso, si $f_{ij} < u_{ij}$ se tiene que $f'_{ij} \leq u_{ij} - f^0_{ij} = u'_{ij}$, por tanto f'_{ij} satisface las restricciones de capacidad. Mientras tanto, si $f_{ij} < f^0_{ij}$ fijamos $f'_{ij} = 0$ y $f'_{ji} = f^0_{ij} - f_{ij}$. En este caso $0 \leq f'_{ji} \leq f^0_{ij} = u'_{ji}$, por tanto f'_{ji} también satisface las restricciones de capacidad del problema. Queda demostrado que si $\boldsymbol{f}$ es un flujo factible en R, entonces el flujo $\boldsymbol{f'}$ correspondiente es un flujo factible en $R^{\boldsymbol{f}^0}$. Análogamente se demuestra que, siendo $\boldsymbol{f'}$ un flujo factible en la red residual $R^{\boldsymbol{f}^0}$, la solución dada por $f_{ij} = (f'_{ij} - f'_{ji}) + f^0_{ij}$ es un flujo factible en R.

Establezcamos pues la relación entre los costes de un flujo $\boldsymbol{f}$ en R y el correspondiente flujo $\boldsymbol{f'}$ en $R^{\boldsymbol{f}^0}$. Para todo arco $(i,j) \in A$, $c'_{ij} = c_{ij}$ y $c'_{ji} = -c_{ij}$. Para un flujo f_{ij} sobre el arco (i,j) de la red original R, el coste del flujo de los arcos (i,j) y (j,i) en la red residual es:

$$c'_{ij} f'_{ij} + c'_{ji} f'_{ji} = c'_{ij}(f'_{ij} - f'_{ji}) = c_{ij} f_{ij} - c_{ij} f^0_{ij},$$

donde la última igualdad se sigue de (2.1). Por tanto queda demostrado que

$$\boldsymbol{c'f'} = \boldsymbol{cf} - \boldsymbol{cf}^0.$$

□

Como consecuencia de esta propiedad, una vez determinada una solución óptima en la red residual podemos convertirla fácilmente en una solución óptima en la red original. Cabe destacar que varios de los algoritmos para el problema del flujo máximo y para el PFCM que presentaremos utilizan este resultado.

2.2. Emparejamientos, caminos aumentadores y flores

En esta sección presentaremos algunos conceptos y resultados sobre teoría de grafos que serán necesarios para el estudio de los problemas de emparejamiento. Estos resultados, que aparecen en el Capítulo 12 de Ahuja et al. (1993), serán especialmente útiles a la hora de estudiar las distintas versiones del problema de emparejamiento no bipartito (Capítulo 5). Por considerar triviales las propiedades que presentamos, sus demostraciones no están incluidas en el trabajo, así como tampoco lo están en la bibliografía consultada.

Arcos y nodos emparejados

Un **emparejamiento** M sobre un grafo $G = (N, A)$ es un subconjunto de arcos, $M \subseteq A$, con la propiedad de que no hay dos arcos en M que sean incidentes en el mismo nodo. Los arcos de M se llaman **arcos emparejados**, mientras que los de $A \setminus M$ son **arcos no emparejados**. Análogamente, N se puede dividir en **nodos emparejados** (nodos incidentes en arcos emparejados) y **nodos no emparejados**. Si $(i, j) \in M$ decimos que el nodo i está emparejado con el nodo j.

Nótese que un emparejamiento en un grafo G con n nodos está formado por, a lo sumo, $\lfloor n/2 \rfloor$ arcos. Se dice que un **emparejamiento** es **máximo** si empareja el mayor número posible de nodos de G.

Dado un emparejamiento $M = \{a_1, \ldots, a_k\} \subseteq A$ formado por k arcos, también podemos referirnos a él mencionando los pares de nodos que están emparejados, es decir, $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\} \subseteq (N \times N)^k$.

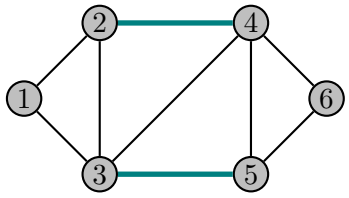


Figura 2.1: En este ejemplo $M = \{(2, 4), (3, 5)\}$ y los nodos 1 y 6 son nodos no emparejados.

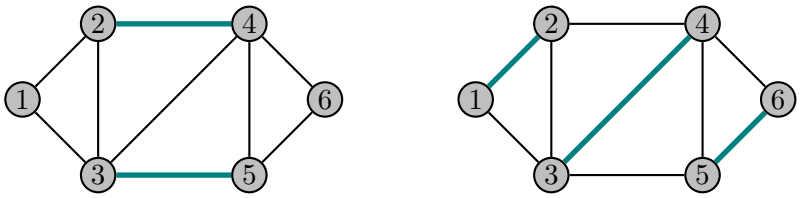
Caminos y ciclos alternados

Sea $P = i_1 \hookrightarrow i_2 \hookrightarrow \ldots \hookrightarrow i_k$ un camino en un grafo G, decimos que P es un **camino alternado** con respecto al emparejamiento M si empieza en un nodo no emparejado y, para cada par de arcos consecutivos, uno pertenece al emparejamiento y el otro no. Además un camino alternado es **par** si está formado por un número par de arcos y es **impar** si

está formado por un número impar de arcos. También podemos hablar de **ciclo alternado** cuando el camino empieza y termina en el mismo nodo.

Caminos aumentadores

Un camino alternado impar P con respecto a un emparejamiento M se dice que es un **camino aumentador**. Nótese que el primer y el último nodo de un camino aumentador no están emparejados, de hecho se llama *aumentador* porque se puede usar para encontrar emparejamientos de mayor cardinalidad que uno dado, sin más que cambiar los arcos emparejados del camino por arcos no emparejados y viceversa.



(a) Emparejamiento M (b) Emparejamiento M'

Figura 2.2: Usando un camino aumentador para encontrar un emparejamiento de mayor cardinalidad.

Por ejemplo, en la figura anterior tenemos un emparejamiento $M = \{(2,4),(3,5)\}$ de cardinalidad 2. En este grafo encontramos fácilmente un camino alternado: $1 \hookrightarrow 2 \hookrightarrow 4 \hookrightarrow 3 \hookrightarrow 5$; además, $P = 1 \hookrightarrow 2 \hookrightarrow 4 \hookrightarrow 3 \hookrightarrow 5 \hookrightarrow 6$ es un camino aumentador, que podemos utilizar para detectar el emparejamiento $M' = \{(1,2),(3,4),(5,6)\}$ de cardinalidad 3. Además, este último emparejamiento es máximo, ya que empareja todos los nodos de N.

Diferencia simétrica

Sean S_1 y S_2 dos conjuntos, su **diferencia simétrica** es: $S_1 \oplus S_2 = (S_1 \cup S_2) - (S_1 \cap S_2)$, es decir, la diferencia simétrica de dos conjuntos está formada por los elementos que están en uno de ellos pero no en ambos. Este concepto nos será de gran utilidad a la hora de estudiar emparejamientos gracias al siguiente resultado.

Propiedad 2.3. *Sean M un emparejamiento sobre un grafo G y P un camino aumentador con respecto a M. Entonces $M \oplus P$ es un emparejamiento de cardinalidad $|M|+1$ sobre G, en el cual todos los nodos que estaban emparejados en M siguen emparejados y se añaden el primer y el último nodo de P.*

Nótese que cuando reemplazamos un emparejamiento por su diferencia simétrica con P lo que hacemos es intercambiar los arcos emparejados y los no emparejados de P. Por ejemplo, en la figura anterior:

$$\begin{cases} M = \{(2,4),(3,5)\}, \\ P = \{(1,2),(2,4),(4,3),(3,5),(5,6)\}, \\ M' = M \oplus P = \{(1,2),(4,3),(5,6)\}. \end{cases}$$

Propiedad 2.4. *Si M y M^* son dos emparejamientos, su diferencia simétrica define el subgrafo $G^* = (N, M \oplus M^*)$ con la propiedad de que cada componente es de uno de los tipos mostrados en la siguiente figura.*

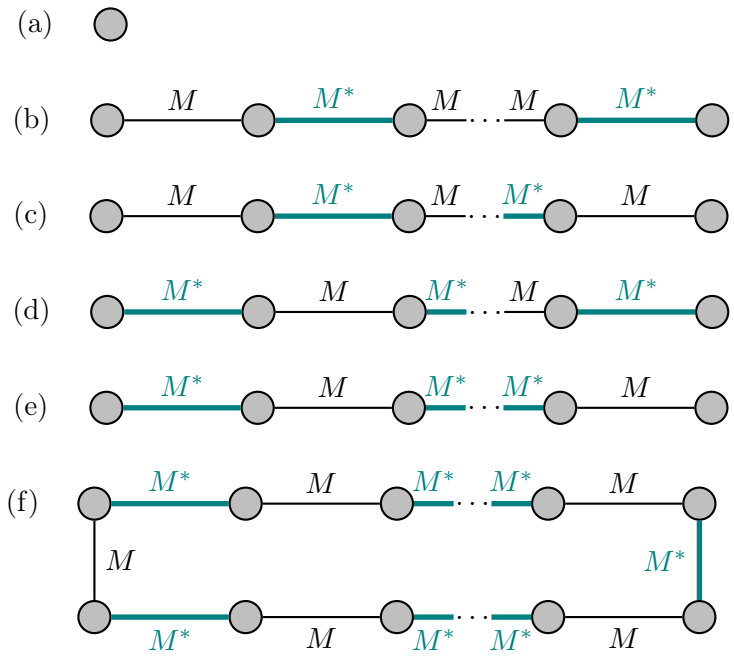


Figura 2.3: Posibles componentes formadas al hacer $M \oplus M^*$.

Esta propiedad se sigue del hecho de que en el subgrafo G^* cada nodo cuenta con 0, 1 o 2 arcos incidentes en él, y las únicas posibles componentes que siguen este esquema son las seis mostradas en la Figura 2.3.

Teorema del camino aumentador

La demostración del siguiente teorema se puede consultar en Berge (1957).

Teorema 2.5. *Un emparejamiento M^* sobre un grafo G es máximo si, y solo si, G no contiene ningún camino aumentador con respecto a M^*.*

Una versión alternativa de este resultado es el siguiente teorema, que será crucial en el desarrollo de los algoritmos para el problema de emparejamiento.

Teorema 2.6. *(Teorema del camino aumentador). Si un nodo p no está emparejado en un emparejamiento M y M no contiene caminos aumentadores que empiecen en p, entonces existe algún emparejamiento máximo sobre G en el cual el nodo p no está emparejado.*

Demostración. Sea M^* un emparejamiento máximo. Si el nodo p es un nodo no emparejado en M^*, claramente el teorema es cierto. Supongamos entonces que p está emparejado en M^*. Si consideramos el subgrafo definido por la diferencia simétrica de ambos emparejamientos, $M \oplus M^*$, sabemos por la propiedad 2.4 que cada componente será del tipo de las mostradas en la Figura 2.3. Dado que, por hipótesis, p no está emparejado en M y no existen caminos aumentadores en M que comiencen en p, nos queda sólamente la posibilidad (e). El camino mostrado en (e), al que llamaremos P, es un camino alternado de longitud par que empieza en p. Nótese que $M' = M^* \oplus P$ es también un emparejamiento máximo en el cual p es un nodo no emparejado. Hemos construido a partir de M^* otro emparejamiento máximo M' donde p es un nodo no emparejado, con lo que queda demostrado el teorema. □

Flores y corolas

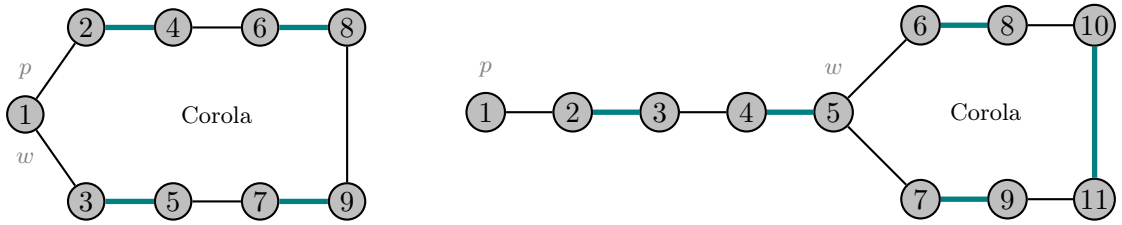
En los problemas de emparejamiento sobre grafos no bipartitos pueden presentarse dificultades por la presencia de ciertos subgrafos llamados flores, por lo que enunciaremos algunas de sus propiedades para hacer uso de ellas en el Capítulo 5.

Dado un emparejamiento M sobre $G = (N, A)$, una **flor** en M con **nodo raíz** p es un subgrafo de G con dos componentes:

- **Tallo**: es un camino alternado de longitud par que empieza en el nodo raíz p y termina en algún nodo w. Se permite que $p = w$, en cuyo caso la flor no tendría tallo.
- **Corola**:[2] es un ciclo alternado de longitud impar, que empieza y termina en el nodo final del tallo, w, y que no tiene ningún otro nodo en común con él. Llamamos **base** de la corola al nodo w.

Nótese que dado que los grafos bipartitos no contienen ciclos de longitud impar, no pueden contener flores.

[2] Denotaremos las corolas con la letra B por la palabra inglesa para este tipo de subgrafos, *blossom*.



(a) Flor sin tallo. (b) Flor con tallo.

Figura 2.4: Dos ejemplos de flores.

Podemos definir una corola a partir del conjunto de arcos o de nodos que la forman, igual que hacíamos con las cadenas, dependiendo de qué sea más conveniente en cada caso.

Propiedad 2.7. *Una flor en un emparejamiento M sobre el grafo $G = (N, A)$ cumple:*

a) Existe un entero $l \geq 0$ tal que el tallo abarca $2l + 1$ nodos y l arcos emparejados.

b) Existe un entero $k \geq 1$ tal que la corola abarca $2k + 1$ nodos y k arcos emparejados, estos arcos emparejan a todos los nodos de la corola excepto al nodo base w.

c) La base de la corola es un nodo Par, es decir, el único camino que va desde el nodo raíz hasta el nodo base (el tallo) es de longitud par.

La siguiente propiedad, aunque pueda parecer intrascendente, conlleva un gran contratiempo a la hora de buscar caminos aumentadores en grafos que contengan flores. Por eso, en el siguiente apartado, explicamos cómo deshacerse de este tipo de subgrafos.

Propiedad 2.8. *Sea M un emparejamiento sobre un grafo $G = (N, A)$ y B una corola en M. Todo nodo i de la corola, excepto su base w, se puede alcanzar desde el nodo raíz p (o desde la propia base w) a través de dos caminos alternados distintos, uno de longitud par y otro de longitud impar. El camino alternado de longitud par llega al nodo i a través de un arco emparejado y el de longitud impar a través de un arco no emparejado.*

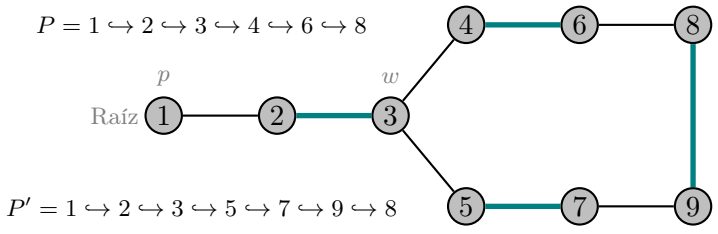


Figura 2.5: Dos caminos alternados con distinta paridad desde p hasta un nodo de la corola.

Contraer una corola

Uno de los algoritmos que presentaremos para resolver el problema de emparejamiento de máxima cardinalidad (que busca emparejamientos máximos sobre un grafo G) se basa en encontrar caminos aumentadores y hacer uso de la Propiedad 2.3 para ir aumentando la cardinalidad del emparejamiento de partida.

Este algoritmo asignará a cada nodo una etiqueta, Par o Impar, dependiendo de la longitud del camino que va desde otro nodo concreto del grafo hasta él. En los grafos que contienen una flor, por la Propiedad 2.8, estas etiquetas no estarán bien definidas. Un mismo nodo puede obtener etiqueta Par o Impar dependiendo del camino alternado por el que lleguemos hasta él desde el nodo raíz p o desde la base de la corola w.

Una de las propuestas más populares para remediar este problema consiste en contraer la corola en un único nodo. Esta operación reemplaza la corola B, que consiste en una secuencia de nodos $i_1 \hookrightarrow i_2 \hookrightarrow \ldots \hookrightarrow i_k \hookrightarrow i_1$, por un nuevo nodo b de la siguiente manera:

1. Introducimos en el grafo un nuevo nodo b y definimos su lista de adyacencia como $A_b = A_{i_1} \cup A_{i_2} \cup \ldots \cup A_{i_k}$.

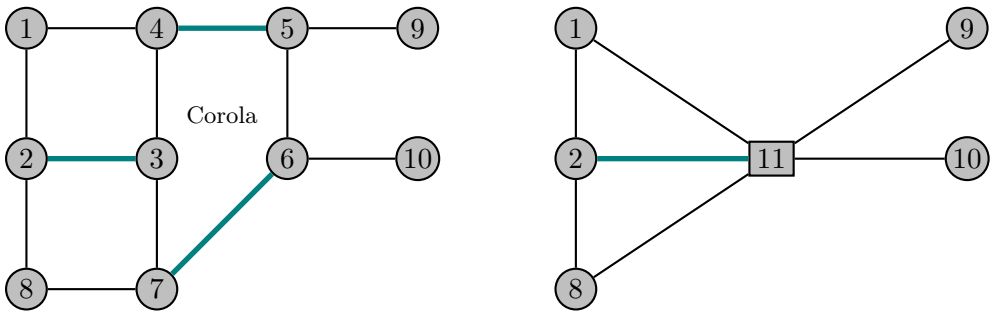
2. Actualizamos la lista de adyacencia de todo nodo $j \in A_b$ haciendo $A_j = A_j \cup \{b\}$.

3. Para poder recuperar posteriormente la información sobre los nodos de la corola contraída, creamos una lista con los nodos que la forman, para después eliminar del grafo los nodos $i_1, i_2, \ldots, i_k$ y los arcos incidentes en ellos. Nótese que esta operación conlleva la actualización de las listas de adyacencia de todos los nodos adyacentes a los eliminados.

Si partimos de un emparejamiento M en un grafo $G = (N, A)$ que contiene una corola, llamaremos **grafo contraído**, $G^c = (N^c, A^c)$, al grafo resultante después de contraer la corola. Además A^c_i será la lista de adyacencia del nodo i en el grafo G^c y M^c el emparejamiento correspondiente en el grafo contraído.

Cada vez que se contrae una corola se crea un nuevo nodo en G. Para diferenciar estos nodos de los nodos del grafo original los llamaremos **pseudo-nodos**.[3] Nótese que un pseudo-nodo será siempre un nodo Par, ya que se podría decir que se encogen todos los nodos de una corola hacia su base, la cual es siempre un nodo Par (por la Propiedad 2.7). Por tanto, contraer una corola en un único pseudo-nodo equivale a asignar etiquetas Par a todos los nodos de G que la forman.

[3] A la hora de dibujar un grafo, los pseudo-nodos se representan mediante rectángulos para de diferenciarlos de los nodos del grafo original.



(a) Grafo G antes de la contracción. (b) Grafo G^c, después de la contracción.

Figura 2.6: Contracción de una corola.

Además es posible contraer corolas que contengan pseudo-nodos, por lo que no nos debe extrañar que cuando expandimos un pseudo-nodo aparezca otro pseudo-nodo.

2.3. Algoritmos de búsqueda

En esta sección trataremos un tipo de algoritmos que se usan bastante a lo largo del trabajo, los llamados algoritmos de búsqueda. El estudio de su orden de convergencia y otros resultados interesantes sobre ellos se pueden encontrar con más detalle en el Capítulo 3 de Ahuja et al. (1993).

Los algoritmos de búsqueda, los cuales intentan encontrar todos los nodos de una red que satisfacen una cierta propiedad, son una herramienta fundamental para el diseño de algoritmos más complejos. Muchos de los algoritmos que presentamos para resolver problemas de flujo máximo (Capítulo 3) o problemas de emparejamiento se apoyan en distintas variantes de algoritmos de búsqueda.

Supongamos que queremos encontrar, en un grafo dirigido $G = (N, A)$, todos los nodos que sean alcanzables a lo largo de caminos dirigidos desde un nodo s, llamado **nodo fuente**. Un algoritmo de búsqueda parte del nodo s y va identificando nodos alcanzables del siguiente modo: en cada iteración otorga a cada nodo de la red uno de los siguientes dos estados: **marcado** o **no marcado**. Los nodos marcados son los que ya se sabe que son alcanzables desde el nodo fuente s, mientras que de los no marcados aún no se tiene información. Si el nodo i está marcado, el nodo j no lo está y el arco (i, j) está contenido en el grafo, entonces podemos automáticamente marcar el nodo j. En este caso nos referiremos al arco (i, j) como un **arco admisible**.

Estos algoritmos pueden aplicarse también a grafos no dirigidos, en tal caso un arco será admisible cuando uno de los nodos incidentes en él esté marcado y el otro no.

Cuando se marca un nuevo nodo j al examinar el arco admisible (i, j), se dice que el nodo i es el **predecesor** del nodo j y se denota $pred_j = i$. Cuando el nodo j pasa a ser un nodo marcado, el algoritmo almacena la información para saber que, cuando termine con el nodo actual i, debe buscar arcos admisibles en la lista de adyacencia A_j.

El algoritmo examina la lista de adyacencia de cada nodo i y marca los nodos alcanzables desde él. Cuando no quedan nodos en A_i declara que el nodo i no tiene arcos admisibles y continúa examinando las listas de adyacencia de los nodos marcados. En general, la manera de examinar los arcos admisibles que salen de un nodo i es en orden creciente de los nodos terminales, es decir, si (i, j) e (i, k) son arcos de A_i y $j < k$ entonces se examina el arco (i, j) y después el (i, k).

Cuando la red no contiene arcos admisibles el algoritmo termina y los predecesores definen un árbol formado por nodos marcados al que llamaremos **árbol de búsqueda**. A continuación se muestran dos árboles de búsqueda relativos al grafo de la Figura 2.7(a). Nótese que el árbol de búsqueda relativo a un grafo no es único, ya que los arcos incluidos en él dependen del orden en el que examinemos los arcos admisibles. De todos modos, un árbol de búsqueda de un grafo siempre incluirá los mismos nodos.

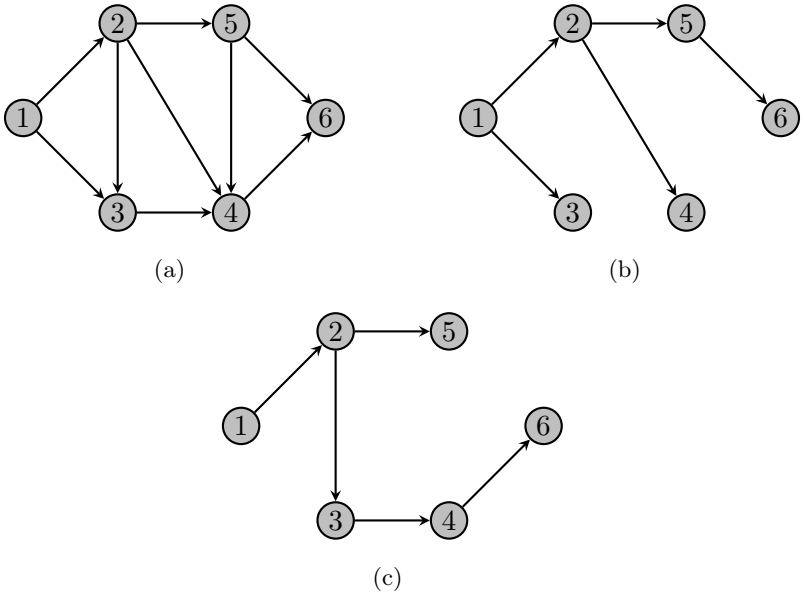


Figura 2.7: Dos árboles de búsqueda de un grafo.

Es fácil demostrar que el algoritmo de búsqueda encuentra un árbol de búsqueda sobre un grafo $G = (N, A)$ en un tiempo de $O(m)$. Pues, si el grafo tiene un total de m arcos, el algoritmo examina cada arco a lo sumo una vez.

2.3.1. Algoritmo de búsqueda de caminos aumentadores

Tal y como describimos el algoritmo de búsqueda genérico, este marca los nodos que son alcanzables desde otro nodo a través de un camino dirigido, pero existen otros muchos criterios para marcar o no marcar un nodo. El criterio que más nos interesa a lo largo de este trabajo, como herramienta para resolver los problemas de emparejamiento, es el de que un nodo sea alcanzable desde otro prefijado a través de un camino aumentador.

A continuación describimos el algoritmo básico de búsqueda de un árbol alternado, el cual será esencial para la resolución de la versión bipartita de los problemas de emparejamiento. Este algoritmo puede fallar cuando se aplica a grafos no bipartitos por la presencia de flores, por lo que en el Capítulo 5 nos centraremos en modificar esta técnica de búsqueda para solventar la situación en grafos más generales.

Partiendo de un grafo bipartito no dirigido $G = (N, A)$ y un emparejamiento M sobre G, queremos encontrar un árbol de búsqueda con raíz en un nodo $p \in N$ que sea no emparejado, al que nos referiremos como **nodo raíz** del árbol, de manera que cada camino desde el nodo p hasta cualquier otro nodo sea un camino alternado. Diremos que un árbol que cumpla esta característica es un **árbol alternado**. En este caso particular, los nodos marcados se dividen en **pares** e **impares**, etiquetas que se les asignará en el transcurso del algoritmo.[4] Un nodo es Par o Impar dependiendo del número de arcos que conformen el (único) camino que lo une con el nodo raíz. Nótese que cuando un nodo no emparejado i distinto del nodo raíz tiene etiqueta Impar, el camino que une el nodo raíz con este nodo es un camino aumentador.

El algoritmo que presentamos, búsqueda(p,found), crea una lista ($\mathcal{L}$) de nodos marcados y los examina uno a uno en orden creciente. Si examina un nodo con etiqueta Par i, el algoritmo analiza su lista de adyacencia y asigna etiqueta Impar a cada nodo j de A_i. Si j no estaba emparejado ya hemos descubierto un camino aumentador, en otro caso añadimos ese nodo a $\mathcal{L}$. Por otra parte, si examina un nodo con etiqueta Impar i el algoritmo busca la única arista $(i, j) \in M$. Si el nodo j no tiene etiqueta, el algoritmo le asigna etiqueta Par y lo añade a $\mathcal{L}$.

El algoritmo termina cuando $\mathcal{L} = \emptyset$ o bien cuando ha asignado etiqueta Impar a un nodo no emparejado, lo que quiere decir que ha encontrado un camino aumentador.

[4] Nos referiremos a los nodos pares con la letra E y a los impares con la letra O debido a las palabras inglesas *even* y *odd*.

Datos: Partimos de un emparejamiento factible M sobre un grafo bipartito no dirigido G y un nodo p no emparejado.
found $\leftarrow$ *false*
desetiquetar todos los nodos
etiquetar el nodo p como Par e inicializar $\mathcal{L} = \{p\}$
mientras $\mathcal{L} \neq \emptyset$ **hacer**
  eliminar un nodo i de $\mathcal{L}$
  si *el nodo i tiene etiqueta Par* **entonces**
    examinar-Par(i,found)
  en otro caso
    examinar-Impar(i,found)
  fin
  si *found* = *true* **entonces**
    return
  fin
fin

---

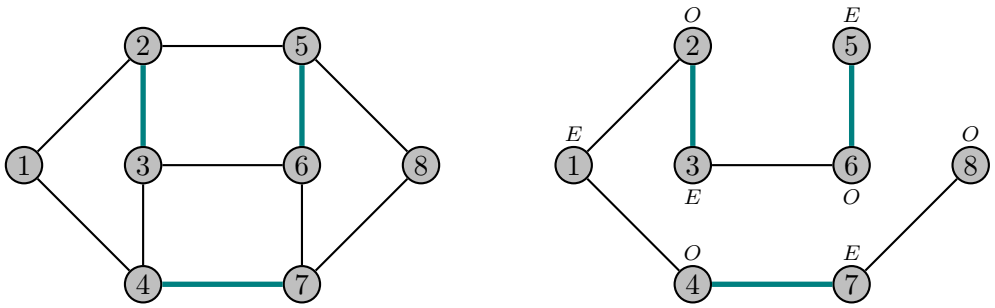
subrutina examinar-Par(i, found)
para *todo nodo* $j \in A_i$ **hacer**
  si j *no está emparejado* **entonces**
    fijar $q \leftarrow j$ y $pred_q \leftarrow i$
    found $\leftarrow$ true
    return
  fin
  si j *está emparejado y no etiquetado* **entonces**
    $pred_j \leftarrow i$
    etiquetar el nodo j como Impar
    añadir el nodo j a $\mathcal{L}$
  fin
fin

---

subrutina examinar-Impar(i, found)
sea j el nodo emparejado al nodo i
si *el nodo j no está etiquetado* **entonces**
  $pred_j \leftarrow i$
  etiquetar el nodo j como Par
  añadir el nodo j a $\mathcal{L}$
fin

Algoritmo 1: Algoritmo de búsqueda(p,found).

Nótese que el algoritmo de búsqueda de caminos aumentadores que comentamos utiliza un argumento auxiliar, found, que toma el valor *true* únicamente si se encuentra un camino aumentador. Este argumento nos servirá de ayuda cuando usemos estos algoritmos como algoritmos auxiliares para el problema de emparejamiento de máxima cardinalidad.



(a) Grafo no dirigido G con nodo raíz $p = 1$ y $M = \{(2,3),(4,7),(5,6)\}$.

(b) Árbol alternado, encontramos el camino aumentador $1 \hookrightarrow 4 \hookrightarrow 7 \hookrightarrow 8$.

Figura 2.8: Árbol alternado identificado con el algoritmo previo.

2.4. Algoritmo de caminos más cortos sucesivos para el PFCM

Este apartado está basado en la Sección 9.7 de Ahuja et al. (1993). El algoritmo de caminos más cortos sucesivos explota las propiedades del problema dual de un PFCM y su relación con el problema original para encontrar el óptimo, por lo que hace uso de los costes reducidos con respecto a unos potenciales de los nodos, $\boldsymbol{c^{\pi}}$. Este algoritmo nos servirá más adelante para resolver el problema de emparejamiento bipartito ponderado o problema de asignación (Sección 4.2) que, como ya vimos, es un caso particular de PFCM.

En cada paso, el algoritmo selecciona un nodo con exceso de suministro, s, otro nodo con demanda insatisfecha, t, y manda unidades de flujo de s hasta t a lo largo del camino más corto entre ellos en la red residual $R^{\boldsymbol{f}}$. Nótese que para elegir el camino más corto con respecto a los costes reducidos, en cada iteración del algoritmo se deben usar algoritmos auxiliares específicos para ese problema, por lo que la velocidad del algoritmo de caminos más cortos sucesivos dependerá de qué algoritmo auxiliar elijamos.

Para facilitar la presentación del siguiente algoritmo supondremos, sin pérdida de generalidad, que partimos de una red con cotas inferiores nulas, véase la Sección 2.1.1. Por tanto, omitiremos las cotas inferiores al representar las redes de flujo y en cada arco aparecerán los parámetros $(u'_{ij}, c^{\boldsymbol{\pi}}_{ij})$.

Prerrequisitos

Un **pseudo-flujo** es una función $\boldsymbol{f} : A \to \mathbb{R}^+$ que no tiene por qué cumplir las restricciones de balance de flujo. Dado un pseudo-flujo $\boldsymbol{f}$ en una red, definimos el **desequilibrio** del nodo $i \in N$ como:

$$e_i = b_i + \sum_{j:(j,i)\in A} f_{ji} - \sum_{j:(i,j)\in A} f_{ij},\ \forall i \in N$$

Si $e_i > 0$ decimos que e_i es el **exceso** del nodo i; si $e_i < 0$ decimos que $-e_i$ es el **déficit** del nodo i y si $e_i = 0$ se dice que i es un nodo **equilibrado**. Por tanto, en R podemos definir E y D como los conjuntos de nodos de la red con exceso y con déficit respectivamente. Nótese que $\sum_{i\in N} e_i = \sum_{i\in N} b_i = 0$, por lo tanto, siempre que una red contenga un nodo con exceso, debe contener algún nodo con déficit. El algoritmo de caminos más cortos sucesivos termina cuando $\boldsymbol{f}$ cumple todas las restricciones de balance de flujo, es decir, cuando todos los nodos están equilibrados, $E = D = \emptyset$.

Partiendo de una red $R = ((N, A), \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b})$ y de un pseudo-flujo $\boldsymbol{f}$ construimos la red residual correspondiente $R^{\boldsymbol{f}}$. En la red $R^{\boldsymbol{f}}$ definimos $\boldsymbol{d}^s = (d_1, d_2, \ldots, d_n)$ como el vector de distancias (costes) de los caminos más cortos desde un nodo fuente s al resto de nodos de la red. Nótese que es en este momento cuando debemos utilizar un algoritmo auxiliar que resuelva el problema del camino más corto, que se ejecutará n veces para buscar el camino más corto entre s y cada nodo de la red.

Observación. Para que el vector $\boldsymbol{d}^s$ esté bien definido debemos poder encontrar un camino entre cualquier par de nodos de la red. Podemos añadir arcos artificiales a la red con costes muy altos y con capacidad ilimitada, que no aparecerán en la solución óptima siempre y cuando el problema original tenga al menos una solución factible.

Sea $C^{\boldsymbol{f}}$ el camino más corto del nodo fuente s a un sumidero t con respecto a los costes reducidos $\boldsymbol{c}^{\boldsymbol{\pi}}$, definimos el siguiente parámetro:[5]

$$\Delta^{\boldsymbol{f}} = \text{mín}\left\{ \min_{(i,j)\in C^{\boldsymbol{f}}} \{u'_{ij}\}, e_s, -e_t \right\},$$

que denota la cantidad máxima de flujo que podemos enviar a través del camino $C^{\boldsymbol{f}}$. Realizamos ahora los siguientes cambios en los flujos:

- Añadimos $\Delta^{\boldsymbol{f}}$ unidades de flujo a los arcos $(i, j) \in A$ que el camino $C^{\boldsymbol{f}}$ recorre en el sentido dado por R.

[5] Deberíamos llamar a este camino $C^{\boldsymbol{f},s,t,\boldsymbol{c}^{\boldsymbol{\pi}}}$ para explicitar que depende del pseudo-flujo, de los nodos fuente y sumidero y de los costes reducidos de la red. Para simplificar la notación lo denotamos por $C^{\boldsymbol{f}}$.

- Restamos $\Delta^{\boldsymbol{f}}$ unidades de flujo a los arcos $(i,j) \in A$ que el camino $C^{\boldsymbol{f}}$ recorre en el sentido contrario al dado por R.

Por tanto, actualizamos el vector de flujos $\boldsymbol{f}$ de manera que incrementamos el flujo del nodo s al nodo t en $\Delta^{\boldsymbol{f}}$ unidades.

Datos: Partimos de una red $R = ((N, A), \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b})$, un pseudo-flujo factible $\boldsymbol{f} = \mathbf{0}$ y un vector de potenciales $\boldsymbol{\pi^0} = \mathbf{0}$.
$e_i \leftarrow b_i$ para todo nodo $i \in N$
inicializar $E = \{i \in N : e_i > 0\}$ y $D = \{i \in N : e_i < 0\}$
mientras $E \neq \emptyset$ **hacer**
  seleccionar un nodo fuente $s \in E$ y un nodo sumidero $t \in D$
  determinar las distancias $\boldsymbol{d}^s$
  identificar el camino más corto $C^{\boldsymbol{f}}$ de s a t
  actualizar los potenciales de los nodos $\boldsymbol{\pi^f} \leftarrow \boldsymbol{\pi^f} - \boldsymbol{d}^s$
  $\Delta^{\boldsymbol{f}} = \text{mín}\{\text{mín}_{(i,j)\in C^{\boldsymbol{f}}}\, u'_{ij}, e_s, -e_t\}$
  aumentar $\Delta^{\boldsymbol{f}}$ unidades de flujo a lo largo de $C^{\boldsymbol{f}}$
  actualizar los desequilibrios $e_s \leftarrow e_s - \Delta^{\boldsymbol{f}}$ y $e_t \leftarrow e_t + \Delta^{\boldsymbol{f}}$
  actualizar $\boldsymbol{f}, R^{\boldsymbol{f}}, E, D$ y los costes reducidos $c^{\boldsymbol{\pi}}_{ij} \leftarrow c_{ij} - \pi^{\boldsymbol{f}}_i + \pi^{\boldsymbol{f}}_j$
fin

Algoritmo 2: Algoritmo de caminos más cortos sucesivos.

En cada iteración los potenciales de los nodos se actualizan en función de la mínima distancia del nodo fuente s a cada uno de ellos. Los costes se van actualizando a medida que lo hacen los potenciales de los nodos, pues cuanto menor sea el coste de un camino en el primal, estaré dispuesta a pagar menos por el trueque de los nodos inicial y final.

En cada iteración el algoritmo resuelve un problema del camino más corto que disminuye el exceso de algún nodo y por tanto el déficit de otro. El algoritmo termina cuando todos los nodos están equilibrados, es decir, cuando se cumplen las n restricciones de balance de flujo, encontrando un óptimo para el PFCM correspondiente.

Por tanto, si llamamos U al valor absoluto del desequilibrio más grande de la red, el algoritmo de caminos más cortos sucesivos encontrará un óptimo en un máximo de nU iteraciones. Si llamamos $S(n, m, C)$ al coste computacional de resolver el problema del camino más corto sin ciclos de longitud negativa (donde C es el mayor coste de la red), el algoritmo terminará en un tiempo $O(nUS(n, m, nC))$. Nótese que en esta última expresión usamos nC en lugar de C, ya que los costes en la red residual están acotados por nC.

Este algoritmo requiere un tiempo pseudo-polinomial para resolver un PFCM. Aunque vimos en la Sección 1.4 que éste no es un tipo de algoritmo muy eficiente, aplicado al problema de emparejamiento bipartito ponderado será más eficiente, ya que para esa clase de problemas se tiene $U = 1$.

2.4.1. Ejemplo de resolución

A continuación resolveremos un problema de flujo en redes a coste mínimo, paso a paso, con el algoritmo de caminos más cortos sucesivos. En cada arco de la red hay dos parámetros, que son $(u'_{ij}, c^{\boldsymbol{\pi}}_{ij})$. Partimos de la siguiente red $R = ((N, A), \boldsymbol{u}, \boldsymbol{c}, \boldsymbol{b}))$

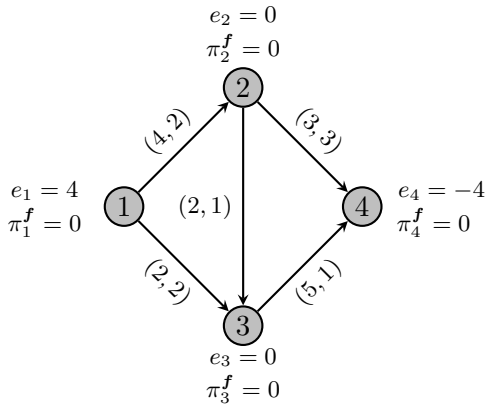
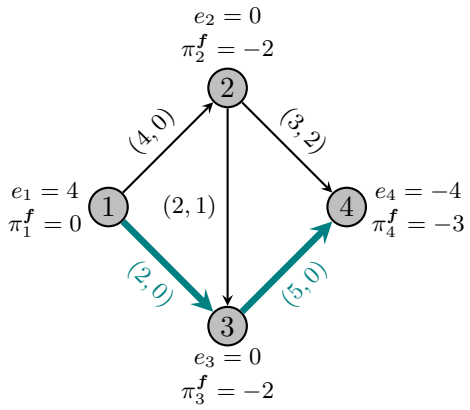


Figura 2.9: Ejemplo de PFCM. Red $R^{\boldsymbol{f}}$ en la iteración I.

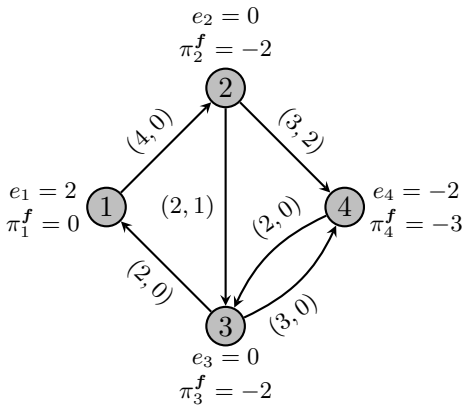
Tenemos un único nodo con exceso, $e_1 = 4$, y un único nodo con déficit, $e_4 = -4$, por tanto buscamos sucesivamente los caminos más cortos para llevar flujo del nodo 1 al 4.

Iteración 1: Partimos de un pseudo-flujo $\boldsymbol{f} = \boldsymbol{0}$ y un vector de potenciales $\boldsymbol{\pi} = \boldsymbol{0}$. Nótese que para estos datos iniciales $R^{\boldsymbol{f}} = R$. Tenemos $E = \{1\}$ y $D = \{4\}$, por tanto $s = 1$ y $t = 4$. Las distancias de los caminos más cortos desde s son $\boldsymbol{d}^s = (0, 2, 2, 3)$ y $C^{\boldsymbol{f}} = 1 \hookrightarrow 3 \hookrightarrow 4$. Actualizamos el vector $\boldsymbol{\pi}^{\boldsymbol{f}}$ y después el vector de costes. En esta iteración $\Delta^{\boldsymbol{f}} = \text{mín}\{u'_{13}, u'_{34}, e_1, -e_4\} = \text{mín}\{2, 5, 4, 4\} = 2$, por tanto incrementamos el flujo del nodo 1 al nodo 4 en dos unidades y actualizamos los desequilibrios e_1 y e_4.

Iteración 2: Partimos de la siguiente red $R^{\boldsymbol{f}}$. Tenemos $E = \{1\}, D = \{4\}$ y por tanto $s = 1$ y $t = 4$. Las distancias de los caminos más cortos desde s son $\boldsymbol{d}^s = (0, 0, 1, 1)$ y $C^{\boldsymbol{f}} = 1 \hookrightarrow 2 \hookrightarrow 3 \hookrightarrow 4$. Actualizamos el vector $\boldsymbol{\pi}^{\boldsymbol{f}}$ y después el vector de costes. En esta iteración $\Delta^{\boldsymbol{f}} = \text{mín}\{u'_{12}, u'_{23}, u'_{34}, e_1, -e_4\} = \text{mín}\{4, 2, 3, 2, 2\} = 2$, por tanto incrementamos el flujo del nodo 1 al nodo 4 en dos unidades y actualizamos los desequilibrios e_1 y e_4.

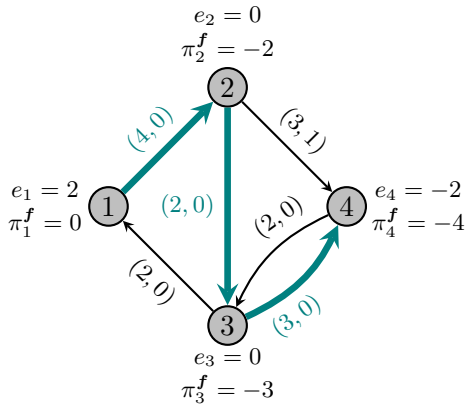


(a) $C^{\boldsymbol{f}}$ en la iteración 1, $\boldsymbol{\pi}^{\boldsymbol{f}}$ y $\boldsymbol{c}^{\boldsymbol{\pi}}$ actualizados.

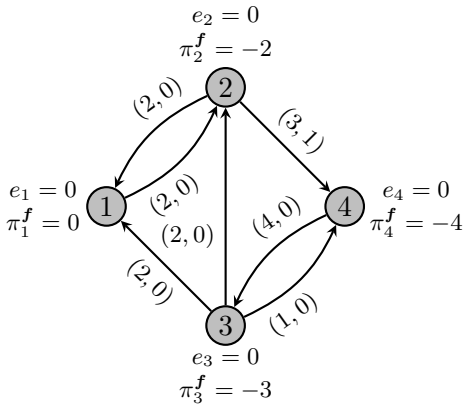


(b) Red $R^{\boldsymbol{f}}$ en la iteración 2.

ITERACIÓN 3: Partimos de la siguiente red $R^{\boldsymbol{f}}$, en la que todos los nodos están equilibrados. En particular $E = \emptyset$, por lo que el algoritmo termina.



(c) $C^{\boldsymbol{f}}$ en la iteración 2, $\boldsymbol{\pi}^{\boldsymbol{f}}$ y $\boldsymbol{c}^{\boldsymbol{\pi}}$ actualizados.



(d) Red $R^{\boldsymbol{f}}$ en la iteración 3.

Obtenemos la siguiente distribución de flujos con coste total 14:

$$f_{12} = 2,\ f_{13} = 2,\ f_{23} = 2,\ f_{34} = 4,$$
$$z = 2c_{12} + 2c_{13} + 2c_{23} + 4c_{34} = 14.$$

Capítulo 3

El problema del flujo máximo

En este capítulo estudiaremos el problema del flujo máximo, al que a veces nos referiremos como PFM. La información sobre los algoritmos que aparecen en la última sección del capítulo, como resultados sobre su convergencia y tiempo de ejecución, se puede ver con más detalle en los Capítulos 6, 7 y 8 de Ahuja et al. (1993). En particular, nos interesa estudiar este problema de optimización porque el problema de emparejamiento bipartito de máxima cardinalidad (Sección 4.1) se puede convertir, tras unos sencillos pasos, en un PFM donde todas las capacidades son unitarias. Por tanto, los algoritmos que presentemos en esta sección para encontrar un flujo máximo en una red, nos servirán más adelante para encontrar un emparejamiento de máxima cardinalidad sobre un grafo bipartito.

Recordemos en qué consiste el problema del flujo máximo: partiendo de una red dirigida y simple, en la que la capacidad de los arcos es limitada, buscamos maximizar la cantidad de flujo que se puede enviar de un nodo fuente s a un nodo sumidero t respetando las capacidades de cada arco. Si denotamos al flujo máximo que buscamos como F, el PFM se puede plantear como un problema de programación lineal como sigue:

$$
\begin{aligned}
\text{Maximizar} \quad & F \\
\text{Sujeto a} \quad & \sum_{\{j:(s,j)\in A\}} f_{sj} - \sum_{\{j:(j,s)\in A\}} f_{js} = F \\
& \sum_{\{j:(i,j)\in A\}} f_{ij} - \sum_{\{j:(j,i)\in A\}} f_{ji} = 0, \quad i \in N, i \neq s, t \\
& \sum_{\{j:(t,j)\in A\}} f_{tj} - \sum_{\{j:(j,t)\in A\}} f_{jt} = -F \\
& 0 \leq f_{ij} \leq u_{ij}, \quad (i,j) \in A.
\end{aligned}
$$

3.1. Algoritmo de trayectorias aumentadas (Ford-Fulkerson)

Esta sección está basada en los apuntes de la asignatura de Programación Lineal y Entera de González-Díaz (2018), a pesar de que el tema en cuestión no se cubre actualmente en la asignatura del Grado en Matemáticas.

El algoritmo de trayectorias aumentadas para el problema del flujo máximo, al que a veces nos referiremos como FFA por sus siglas en inglés, es muy similar el algoritmo de caminos más cortos sucesivos para el PFCM. Este hecho no es de extrañar ya que el problema del flujo máximo se puede ver como un caso particular del PFCM.

Dada una red $R = ((N, A), \boldsymbol{u})$ y un vector de flujos factibles $\boldsymbol{f}$, se construye la red residual $R^{\boldsymbol{f}} = ((N', A'), \boldsymbol{u'})$ como se explicó en la Sección 2.1.4.[1] En $R^{\boldsymbol{f}}$ llamamos **trayectoria aumentada** a cualquier camino dirigido desde el nodo fuente s hasta el sumidero t. El FFA consiste en ir identificando sucesivamente trayectorias aumentadas en $R^{\boldsymbol{f}}$ y actualizando el flujo $\boldsymbol{f}$ hasta que no exista ningún camino dirigido entre la fuente y el sumidero, momento en el cual el algoritmo termina.

Supongamos que en la red $R^{\boldsymbol{f}}$ existe una trayectoria aumentada $C^{\boldsymbol{f}}$,[2] en ese caso podemos conseguir un nuevo flujo mayor al anterior. Sea $\Delta^{\boldsymbol{f}}$ el mínimo cambio de flujo permitido sobre el camino $C^{\boldsymbol{f}}$, es decir:

$$\Delta^{\boldsymbol{f}} = \min_{(i,j)\in C^{\boldsymbol{f}}} \{u'_{ij}\} > 0,$$

hacemos los siguientes cambios en el vector de flujos $\boldsymbol{f}$, de manera que el flujo que va de s a t aumenta en $\Delta^{\boldsymbol{f}}$ unidades.

- Añadimos $\Delta^{\boldsymbol{f}}$ unidades de flujo a los arcos $(i, j) \in A$ que el camino $C^{\boldsymbol{f}}$ recorre en el sentido dado por R.
- Restamos $\Delta^{\boldsymbol{f}}$ unidades de flujo a los arcos $(i, j) \in A$ que el camino $C^{\boldsymbol{f}}$ recorre en el sentido contrario al dado por R.

Además, cuando trabajamos en redes con capacidades finitas este algoritmo encuentra un flujo máximo óptimo en un tiempo limitado por $O(mF)$, donde m es el número de arcos que componen el grafo y F es el flujo máximo. Este límite es debido a que, en cada iteración del algoritmo se encuentra una trayectoria aumentada en un tiempo $O(m)$, que incrementa el valor del flujo en al menos una unidad, es decir, el algoritmo termina en un máximo de F iteraciones.

[1] Recordemos que se puede suponer $l_{ij} = 0, \forall (i, j) \in A$ y que en el PFM no nos preocupan los costes.

[2] Deberíamos llamar a este camino $C^{\boldsymbol{f},\boldsymbol{s},\boldsymbol{t}}$ para explicitar que depende del flujo y de los nodos fuente y sumidero. Para simplificar la notación lo denotamos simplemente por $C^{\boldsymbol{f}}$.

Datos: Partimos de un vector de flujo factible $\boldsymbol{f} = \mathbf{0}$ y una red $R = ((N, A), \boldsymbol{u})$.
mientras $R^{\boldsymbol{f}}$ *contiene un camino dirigido de* s *a* t **hacer**
  identificar un camino $C^{\boldsymbol{f}}$ de s a t
  $\Delta^{\boldsymbol{f}} := \min_{(i,j)\in C^{\boldsymbol{f}}}\{u'_{ij}\}$
  aumentar $\Delta^{\boldsymbol{f}}$ unidades de flujo a lo largo de $C^{\boldsymbol{f}}$ y actualizar $R^{\boldsymbol{f}}$
fin

Algoritmo 3: Algoritmo de trayectorias aumentadas (Ford-Fulkerson).

3.1.1. Ejemplo de resolución

A continuación ilustraremos la resolución de un problema del flujo máximo usando el algoritmo de trayectorias aumentadas. Partimos de la siguiente red con flujos:

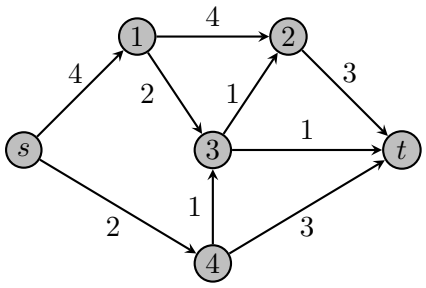
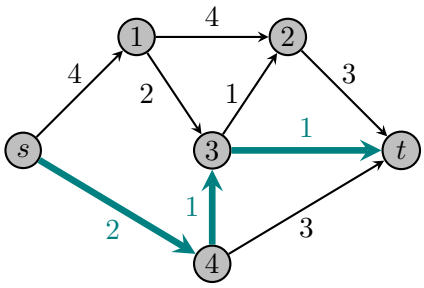


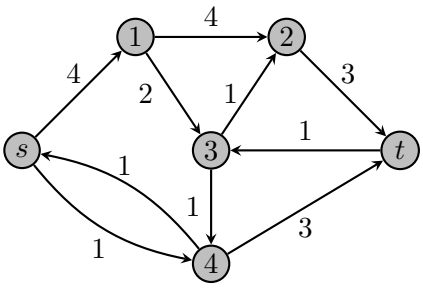
Figura 3.1: Ejemplo de PFM. Red $R^{\boldsymbol{f}}$ en la iteración 1.

ITERACIÓN 1: Partimos de un vector de flujos $\boldsymbol{f} = \mathbf{0}$, con flujo asociado $F = 0$. Para este vector de flujos $R = R^{\boldsymbol{f}}$. Encontramos la trayectoria aumentada $C^{\boldsymbol{f}} = s \hookrightarrow 4 \hookrightarrow 3 \hookrightarrow t$, con $\Delta^{\boldsymbol{f}} = \min\{u'_{s4}, u'_{43}, u'_{3t}\} = \min\{2, 1, 1\} = 1$. Actualizamos las siguientes componentes de $\boldsymbol{f}$: $f_{s4} = 0 + 1 = 1, f_{43} = 0 + 1 = 1$ y $f_{3t} = 0 + 1 = 1$, de modo que el nuevo flujo es $F = 0 + 1 = 1$.

ITERACIÓN 2: Para este vector de flujos tenemos la siguiente red $R^{\boldsymbol{f}}$



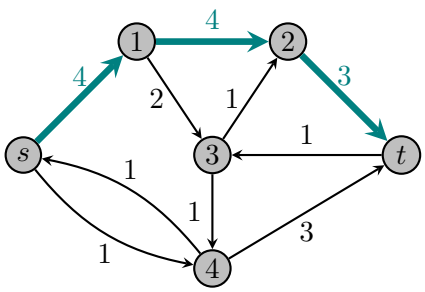
(a) Camino $C^{\boldsymbol{f}}$ en la iteración 1



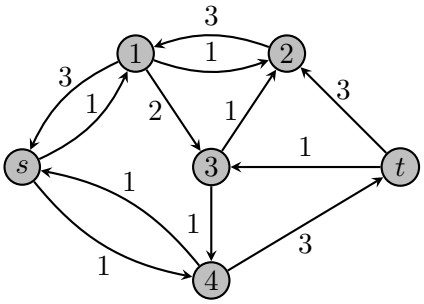
(b) Red $R^{\boldsymbol{f}}$ en la iteración 2

Encontramos la trayectoria aumentada $C^{\boldsymbol{f}} = s \hookrightarrow 1 \hookrightarrow 2 \hookrightarrow t$, con $\Delta^{\boldsymbol{f}} = \text{mín}\{u'_{s1}, u'_{12}, u'_{2t}\} = \text{mín}\{4, 4, 3\} = 3$. Actualizamos las siguientes componentes de $\boldsymbol{f}$: $f_{s1} = 0 + 3 = 3$, $f_{12} = 0 + 3 = 3$ y $f_{2t} = 0 + 3 = 3$, de modo que el nuevo flujo es $F = 1 + 3 = 4$.

ITERACIÓN 3: Para este vector de flujos tenemos la siguiente red $R^{\boldsymbol{f}}$



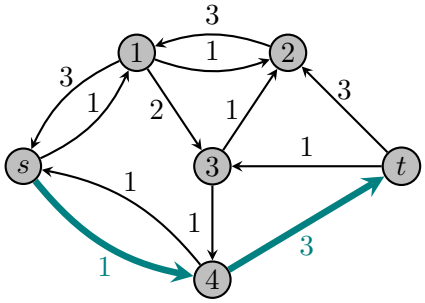
(c) Camino $C^{\boldsymbol{f}}$ en la iteración 2



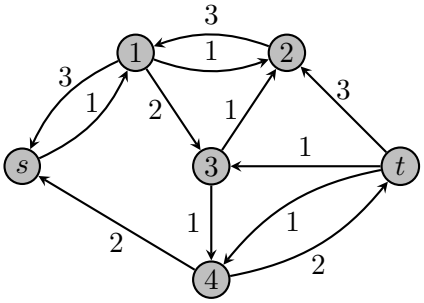
(d) Red $R^{\boldsymbol{f}}$ en la iteración 3

Encontramos las trayectoria aumentada $C^{\boldsymbol{f}} = s \hookrightarrow 4 \hookrightarrow t$, con $\Delta^{\boldsymbol{f}} = \text{mín}\{u'_{s4}, u'_{4t}\} = \text{mín}\{1, 3\} = 1$. Actualizamos las siguientes componentes de $\boldsymbol{f}$: $f_{s4} = 1 + 1 = 2$ y $f_{4t} = 0 + 1 = 1$, de modo que el nuevo flujo es $F = 4 + 1 = 5$.

ITERACIÓN 4: Para este vector de flujos tenemos la siguiente red $R^{\boldsymbol{f}}$



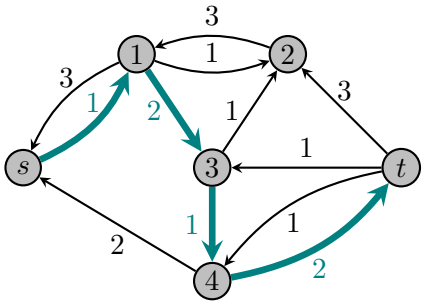
(e) Camino $C^{\boldsymbol{f}}$ en la iteración 3



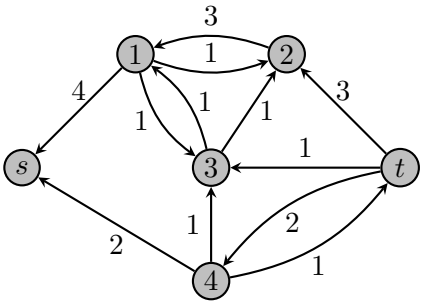
(f) Red $R^{\boldsymbol{f}}$ en la iteración 4

Encontramos la trayectoria aumentada $C^{\boldsymbol{f}} = s \hookrightarrow 1 \hookrightarrow 3 \hookrightarrow 4 \hookrightarrow t$, con $\Delta^{\boldsymbol{f}} = \text{mín}\{u'_{s1}, u'_{13}, u'_{34}, u'_{4t}\} = \text{mín}\{1, 2, 1, 2\} = 1$. Actualizamos las siguientes componentes de $\boldsymbol{f}$: $f_{s1} = 1 + 1 = 2$, $f_{13} = 0 + 1 = 1$, $f_{34} = 1 + 1 = 2$ y $f_{4t} = 0 + 1 = 1$, de modo que el nuevo flujo es $F = 5 + 1 = 6$.

ITERACIÓN 5: Para este vector de flujos tenemos la siguiente red $R^{\boldsymbol{f}}$



(g) Camino C^f en la iteración 4



(h) Red R^f en la iteración 5

En esta última red no existe ninguna trayectoria aumentada de s a t, por lo que el algoritmo termina y obtenemos un flujo máximo de $F = 6$. La distribución de los flujos obtenida (que no tiene por qué ser única) es la siguiente:

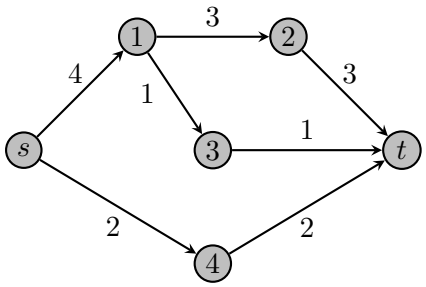


Figura 3.2: Resolución del PFM, $F = 6$.

3.2. PFM en redes con capacidades unitarias

Existen algoritmos específicos para resolver el problema del flujo máximo cuando se trata de una red simple con capacidades unitarias, es decir, donde $u_{ij} = 1$ para todo arco (i, j) de la red. Normalmente, es posible resolver problemas de flujo en redes con capacidades unitarias más eficientemente que en redes generales.

Partiendo del algoritmo de etiquetado y del algoritmo de trayectorias aumentadas más cortas, los cuales tienen cada uno un tiempo de $O(nm)$ cuando se aplican a redes unitarias, desarrollaremos el **algoritmo del flujo máximo para capacidades unitarias**. Este algoritmo es un híbrido de los dos mencionados y consigue un mejor tiempo de ejecución que cualquiera de ellos, ya que encuentra un flujo máximo en un tiempo de $O(\sqrt{n}m)$. Nótese que este algoritmo mejora el tiempo de computación del algoritmo de Ford-Fulkerson de $O(mF)$ hasta un $O(\sqrt{n}m)$.

Este algoritmo consta de dos fases: en la primera aplica el algoritmo de trayectorias aumentadas más cortas para encontrar un flujo *casi óptimo* y en la segunda aplica el algoritmo de etiquetado para convertirlo en un flujo máximo. La primera fase termina cuando se satisface la condición $d_s \geq \text{mín}\{\lceil 2n^{2/3} \rceil, \lceil m^{1/2} \rceil\}$, donde d_s es la distancia del nodo fuente al nodo sumidero con la que trabaja el algoritmo de trayectorias aumentadas más cortas.

A continuación describimos los dos algoritmos en los que se basa el algoritmo del flujo máximo para capacidades unitarias.

3.2.1. Algoritmo de etiquetado

Mientras que el algoritmo de trayectorias aumentadas no explicita una manera de elegir por qué camino enviamos flujo desde la fuente al sumidero en cada iteración, el algoritmo de etiquetado proporciona una manera específica para elegir $C^{\boldsymbol{f}}$. Ya que una vez encontrado ese camino la manera de actualizar los flujos $\boldsymbol{f}$ y la red $R^{\boldsymbol{f}}$ es la misma que en el anterior algoritmo, explicaremos únicamente cómo busca $C^{\boldsymbol{f}}$ el algoritmo de etiquetado.

Nótese que el algoritmo de etiquetado utiliza una técnica de búsqueda para identificar un camino dirigido en $R^{\boldsymbol{f}}$ desde s hasta t como las descritas en la Sección 2.3.

El algoritmo de etiquetado empieza en el nodo fuente y se aleja de él, identificando todos los nodos que son alcanzables desde la fuente a través de un camino orientado de la red residual, con el fin de llegar a alcanzar el nodo sumidero. En cada paso se clasifican los nodos de la red en dos grupos: *etiquetados* ($\mathcal{E}$) y *no etiquetados*. Los etiquetados son aquellos nodos que el algoritmo ha alcanzado en el proceso de búsqueda, por lo que ya ha determinado un camino dirigido en la red residual desde el nodo fuente hasta ellos. El algoritmo de etiquetado selecciona en cada iteración uno de los nodos etiquetados y examina su lista de adyacencia para alcanzar y etiquetar nuevos nodos. Además, se guarda la información de los nodos alcanzados que faltan por examinar en una lista ($\mathcal{L}$), es decir, una vez que se etiqueta un nodo, se incluye en $\mathcal{L}$ hasta que se examine su lista de adyacencia.

Cuando el sumidero entra en el grupo de los etiquetados, el algoritmo envía la máxima cantidad de flujo posible ($\Delta^{\boldsymbol{f}}$) a través del camino construido de s a t usando los predecesores. Después borra todas las etiquetas, actualiza la red $R^{\boldsymbol{f}}$ y empieza de nuevo el proceso. El algoritmo termina cuando, después de examinar las listas de adyacencia de todos los nodos etiquetados, el sumidero sigue sin estar etiquetado, es decir, cuando no existe en la red residual ningún camino dirigido entre el nodo fuente y el sumidero.

Nótese que cuando usemos este algoritmo para resolver un problema de emparejamiento bipartito de máxima cardinalidad partiremos de una red con capacidades unitarias, por lo que $\Delta^{\boldsymbol{f}}$ será siempre igual a 1.

Datos: Partimos de un flujo factible $\boldsymbol{f} = \mathbf{0}$ y una red sin costes $R = ((N, A), \boldsymbol{u})$.
Etiquetar el nodo t
mientras t *esté etiquetado* **hacer**
  desetiquetar todos los nodos
  etiquetar el nodo s y establecer $\mathcal{L} = \{s\}$
  mientras $\mathcal{L} \neq \emptyset$ *y* t *no está etiquetado* **hacer**
    eliminar un nodo i de $\mathcal{L}$
    para *cada arco* $(i, j) \in A'$ **hacer**
      si j *no está etiquetado* **entonces**
        fijar $pred_j \leftarrow i$
        etiquetar j
        añadir el nodo j a $\mathcal{L}$
      fin
    fin
  fin
  si t *está etiquetado* **entonces**
    subrutina aumentar
  fin
fin

---

subrutina aumentar
identificar $C^{\boldsymbol{f}}$ de s a t usando los predecesores
$\Delta^{\boldsymbol{f}} := \min_{(i,j)\in C^{\boldsymbol{f}}}\{u'_{ij}\}$
aumentar $\Delta^{\boldsymbol{f}}$ unidades de flujo a lo largo de $C^{\boldsymbol{f}}$ y actualizar $R^{\boldsymbol{f}}$

Algoritmo 4: Algoritmo de etiquetado.

Este algoritmo sugiere varios resultados importantes, enunciamos dos a continuación.

Teorema 3.1. *(Teorema de trayectorias aumentadas). Un flujo* $\boldsymbol{f}$ *es máximo si, y solo si, la red residual* $R^{\boldsymbol{f}}$ *no contiene trayectorias aumentadas.*

Teorema 3.2. *Si todas las capacidades de una red son enteras, entonces el PFM asociado tiene un flujo máximo entero.*

Si todas las capacidades son enteras existe un entero U que es cota superior de todas ellas, por lo que el valor del flujo máximo está acotado por nU. Cada vez que el algoritmo de etiquetado realiza un aumento incrementa en flujo en, al menos, una unidad. Por ello, en un máximo de nU iteraciones (o aumentos) el algoritmo encuentra un flujo máximo. Además, cada vez que se hace un aumento se ejecuta un algoritmo de búsqueda para formar la trayectoria aumentada y, como vimos en la Sección 2.3, este tipo de algoritmos tienen un tiempo de ejecución de $O(m)$. Por tanto, trabajando en redes con capacidades enteras, el algoritmo de etiquetado encuentra un flujo máximo óptimo en un tiempo limitado por $O(nmU)$, donde U es la mayor capacidad de los arcos del grafo.

Aplicado a redes con capacidades unitarias, podemos concluir que el algoritmo de etiquetado es un algoritmo polinomial y encuentra un flujo máximo en un tiempo de orden $O(nm)$.

Ejemplo de resolución

Veamos un ejemplo de resolución de un PFM usando el algoritmo de etiquetado. Partimos de la siguiente red sin costes $R = ((N, A), \boldsymbol{u})$, donde $s = 1$ y $t = 5$, con flujo $\boldsymbol{f} = \mathbf{0}$.

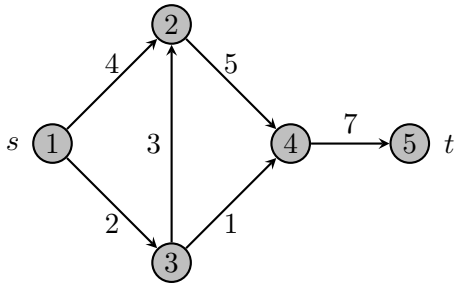
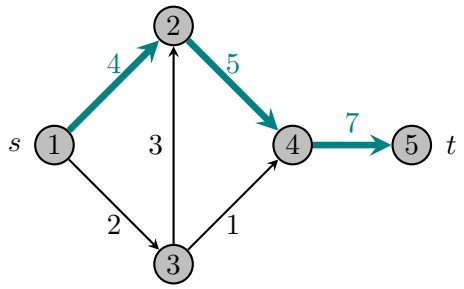


Figura 3.3: Ejemplo de PFM. Red $R^{\boldsymbol{f}}$ en la iteración 1.

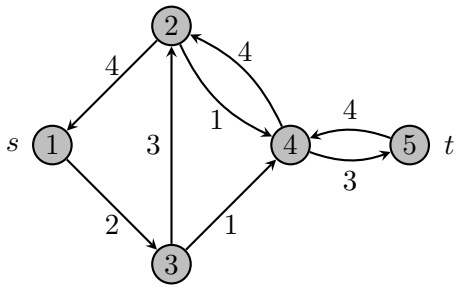
Inicialización: Inicializamos $\mathcal{E} = \{5\}$.

Iteración 1: Como $5 \in \mathcal{E}$, desetiquetamos 5 y establecemos $\mathcal{E} = \{1\}$ y $\mathcal{L} = \{1\}$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 1 de $\mathcal{L}$ e incluímos los nodos 2 y 3. Por tanto: $\mathcal{E} = \{1, 2, 3\}$, $\mathcal{L} = \{2, 3\}$ y $pred_2 = pred_3 = 1$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 2 de $\mathcal{L}$ e incluímos el nodo 4. Por tanto: $\mathcal{E} = \{1, 2, 3, 4\}$, $\mathcal{L} = \{3, 4\}$ y $pred_4 = 2$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 3 de $\mathcal{L}$ y no incluimos ningún nodo, pues los nodos alcanzables desde 3 ya están todos etiquetados. Por tanto: $\mathcal{E} = \{1, 2, 3, 4\}$ y $\mathcal{L} = \{4\}$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 4 de $\mathcal{L}$ e incluímos el nodo 5. Por tanto: $\mathcal{E} = \{1, 2, 3, 4, 5\}$, $\mathcal{L} = \{5\}$ y $pred_5 = 4$.

Como $5 \in \mathcal{E}$, el algoritmo detecta el camino para enviar flujo de s a t usando los predecesores $C^{\boldsymbol{f}} = 1 \hookrightarrow 2 \hookrightarrow 4 \hookrightarrow 5$, envía por él $\Delta^{\boldsymbol{f}} = 4$ unidades de flujo y actualiza la red.

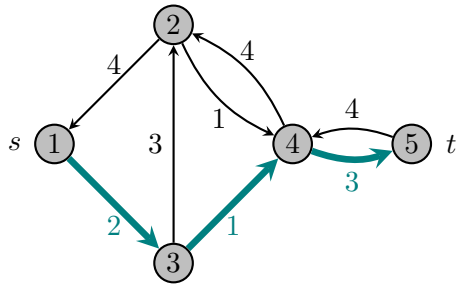


(a) $C^{\boldsymbol{f}}$ encontrado con el algoritmo de etiquetado.

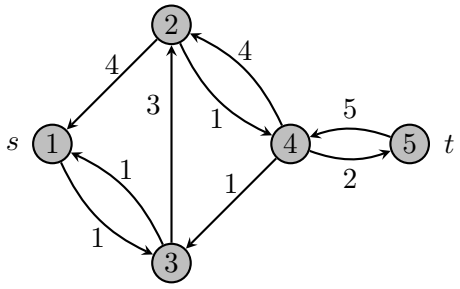


(b) Red $R^{\boldsymbol{f}}$ en la iteración 2.

ITERACIÓN 2: Como $5 \in \mathcal{E}$, desetiquetamos todos los nodos y establecemos $\mathcal{E} = \{1\}$ y $\mathcal{L} = \{1\}$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 1 de $\mathcal{L}$ e incluímos el nodo 3. Por tanto: $\mathcal{E} = \{1, 3\}$, $\mathcal{L} = \{3\}$ y $pred_3 = 1$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 3 de $\mathcal{L}$ e incluímos los nodos 2 y 4. Por tanto: $\mathcal{E} = \{1, 2, 3, 4\}$, $\mathcal{L} = \{2, 4\}$ y $pred_2 = pred_4 = 3$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 2 de $\mathcal{L}$ y no incluimos ningún nodo, pues los nodos alcanzables desde 2 ya están todos etiquetados. Por tanto: $\mathcal{E} = \{1, 2, 3, 4\}$ y $\mathcal{L} = \{4\}$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 4 de $\mathcal{L}$ e incluímos el nodo 5. Por tanto: $\mathcal{E} = \{1, 2, 3, 4, 5\}$, $\mathcal{L} = \{5\}$ y $pred_5 = 4$.
Como $5 \in \mathcal{E}$, el algoritmo detecta el camino para enviar flujo de s a t usando los predecesores, $C^{\boldsymbol{f}} = 1 \hookrightarrow 3 \hookrightarrow 4 \hookrightarrow 5$, envía por él $\Delta^{\boldsymbol{f}} = 1$ unidad de flujo y actualiza la red.



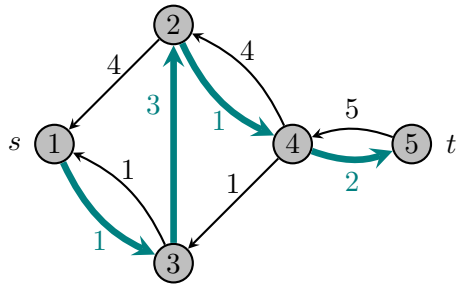
(c) $C^{\boldsymbol{f}}$ encontrado con el algoritmo de etiquetado.



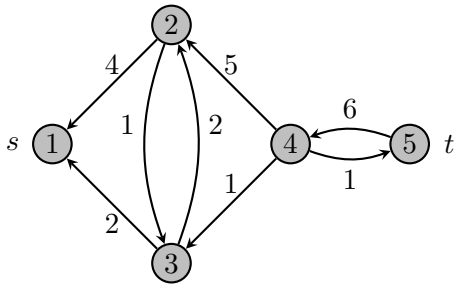
(d) Red $R^{\boldsymbol{f}}$ en la iteración 3.

ITERACIÓN 3: Como $5 \in \mathcal{E}$, desetiquetamos todos los nodos y establecemos $\mathcal{E} = \{1\}$ y $\mathcal{L} = \{1\}$.

Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 1 de $\mathcal{L}$ e incluimos el nodo 3. Por tanto: $\mathcal{E} = \{1, 3\}$, $\mathcal{L} = \{3\}$ y $pred_3 = 1$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 3 de $\mathcal{L}$ e incluimos el nodo 2. Por tanto: $\mathcal{E} = \{1, 2, 3\}$, $\mathcal{L} = \{2\}$ y $pred_2 = 3$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 2 de $\mathcal{L}$ e incluimos el nodo 4. Por tanto: $\mathcal{E} = \{1, 2, 3, 4\}$, $\mathcal{L} = \{4\}$ y $pred_4 = 2$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 4 de $\mathcal{L}$ e incluimos el nodo 5. Por tanto: $\mathcal{E} = \{1, 2, 3, 4, 5\}$, $\mathcal{L} = \{5\}$ y $pred_5 = 4$.
Como $5 \in \mathcal{E}$, el algoritmo detecta el camino para enviar flujo de s a t usando los predecesores, $C^{\boldsymbol{f}} = 1 \hookrightarrow 3 \hookrightarrow 2 \hookrightarrow 4 \hookrightarrow 5$, envía por él $\Delta^{\boldsymbol{f}} = 1$ unidad de flujo y actualiza la red.



(e) $C^{\boldsymbol{f}}$ encontrado con el algoritmo de etiquetado.



(f) Red $R^{\boldsymbol{f}}$ en la iteración 4.

Iteración 4: Como $5 \in \mathcal{E}$, desetiquetamos todos los nodos y establecemos $\mathcal{E} = \{1\}$ y $\mathcal{L} = \{1\}$.
Como $\mathcal{L} \neq \emptyset$ y $5 \notin \mathcal{E}$, eliminamos el nodo 1 de $\mathcal{L}$ y examinamos su lista de adyacencia, que en este caso es vacía. Como $\mathcal{L} = \emptyset$ y el nodo sumidero no está etiquetado el algoritmo termina, obteniendo un flujo $F = 6$, que sigue la siguiente distribución en la red R:

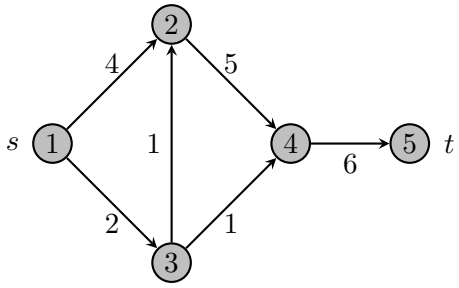


Figura 3.4: Flujo máximo encontrado, $F = 6$.

3.2.2. Algoritmo de trayectorias aumentadas más cortas

Al igual que el algoritmo de etiquetado, el algoritmo de trayectorias aumentadas más cortas proporciona una manera específica para encontrar los caminos $C^{\boldsymbol{f}}$ que se buscan en el FFA. En cada iteración $C^{\boldsymbol{f}}$ será la trayectoria aumentada de mínima longitud, referida al número de arcos. Ya que una vez encontrado ese camino la manera de actualizar los flujos $\boldsymbol{f}$ y la red $R^{\boldsymbol{f}}$ es la misma que en el algoritmo presentado en la Sección 3.1, explicaremos únicamente cómo busca $C^{\boldsymbol{f}}$ el algoritmo de trayectorias aumentadas más cortas.

Sea R una red con grafo subyacente $G = (N, A)$, definiremos el vector de distancias $\boldsymbol{d}^t = (d_1, \ldots, d_n)$ de la siguiente manera: a cada nodo $i \in N$ se le asocia un número que se corresponde con la mínima distancia, referida al número de arcos, entre el sumidero t y él. Un arco $(i, j) \in A'$ se dirá que es un **arco admisible** si $d_i = d_j + 1$. Un camino de s a t será un **camino admisible** si está formado enteramente por arcos admisibles.

El algoritmo de trayectorias aumentadas más cortas busca el camino $C^{\boldsymbol{f}}$, que tendrá que ser un camino admisible, de la siguiente manera: partiendo del nodo fuente s toma un arco admisible que parta de él, cuando hay varios escoge por convenio el arco (s, j) con menor valor de j. Se añade este arco al camino y se guarda la información usando predecesores, en este caso $pred_j = s$. Desde el nodo j se busca otro arco admisible y así sucesivamente, hasta que se alcance el nodo sumidero t. Una vez alcanzado el sumidero obtenemos la trayectoria aumentada $C^{\boldsymbol{f}}$ haciendo uso de los predecesores, enviamos la máxima cantidad de flujo posible a través de ella ($\Delta^{\boldsymbol{f}}$), actualizamos la red $R^{\boldsymbol{f}}$ y se empieza de nuevo.

A continuación presentamos una condición suficiente para que no exista ninguna trayectoria aumentada por la cual enviar más unidades de flujo de s a t, cuya demostración se puede consultar en la Sección 7.2 de Ahuja et al. (1993).

Propiedad 3.3. *Si $d_s \geq n$, la red residual no contiene caminos dirigidos entre el nodo fuente s y el sumidero t.*

El algoritmo termina cuando en la red residual no existen trayectorias aumentadas, es decir, cuando la etiqueta de distancia del nodo fuente satisface $d_s \geq n$ o cuando del nodo fuente no sale ningún arco, $A_s = \emptyset$. Así, por el Teorema 3.1, tenemos asegurado que el flujo que obtiene el algoritmo de trayectorias aumentadas más cortas es máximo.

Nótese que el vector $\boldsymbol{d}^t$ solamente sufre cambios cuando nos encontramos con un nodo "del que no podemos salir". Cuando llegamos a un nodo $i \in N$ del que no sale ningún arco admisible hacemos el siguiente cambio: $d_i = \text{mín}\{d_j + 1 : (i, j) \in A'\}$, lo que hace que el arco $(pred_i, i)$ se convierta en inadmisible. En este caso retrocedemos en el camino y volvemos a implementar el algoritmo, partiendo ahora del nodo $pred_i$.

Datos: Partimos de un flujo factible $\boldsymbol{f} = \mathbf{0}$ y una red sin costes $R = ((N, A), \boldsymbol{u})$.
Determinar el vector de distancias $\boldsymbol{d}^t$
Hacemos $i \leftarrow s$
mientras $d_s < n$ *y* $A_s \neq \emptyset$ **hacer**
  si i *tiene un arco admisible* **entonces**
    subrutina avanzar(i)
    si $i = t$ **entonces**
      subrutina aumentar y fijar $i \leftarrow s$
    fin
  en otro caso
    subrutina actualizar(i)
  fin
fin

---

subrutina avanzar(i)
sea (i, j) un arco admisible de A'
$pred_j \leftarrow i$ y $i \leftarrow j$

---

subrutina actualizar(i)
$d_i \leftarrow \text{mín}\{d_j + 1 : (i, j) \in A'\}$
si $i \neq s$ **entonces**
  $i \leftarrow pred_i$
fin

---

subrutina aumentar
identificar $C^{\boldsymbol{f}}$ de s a t usando los predecesores
$\Delta^{\boldsymbol{f}} := \text{mín}_{(i,j) \in C^{\boldsymbol{f}}}\{u'_{ij}\}$
aumentar $\Delta^{\boldsymbol{f}}$ unidades de flujo a lo largo de $C^{\boldsymbol{f}}$ y actualizar $R^{\boldsymbol{f}}$

Algoritmo 5: Algoritmo de trayectorias aumentadas más cortas.

En la Sección 7.4 de Ahuja et al. (1993) se puede encontrar la demostración del siguiente resultado:

Teorema 3.4. *El algoritmo de trayectorias aumentadas más cortas tiene un tiempo de ejecución de orden* $O(n^2 m)$.

Ejemplo de resolución

Dado que estamos estudiando este algoritmo como una herramienta para resolver el problema de emparejamiento bipartito de máxima cardinalidad (Sección 4.1), ilustraremos su funcionamiento sobre una red del estilo de las que obtendremos cuando estudiemos ese caso particular del problema de emparejamiento.

Nótese que el número asociado a cada nodo es su distancia al sumidero, d_i, y que los arcos no admisibles los representamos punteados. Cuando usemos este algoritmo para resolver un problema de emparejamiento bipartito de máxima cardinalidad todas las capacidades de la red serán igual a 1 y $\Delta^{\boldsymbol{f}} = 1$, por lo que en este ejemplo supondremos lo mismo. Partimos de la siguiente red con flujos:

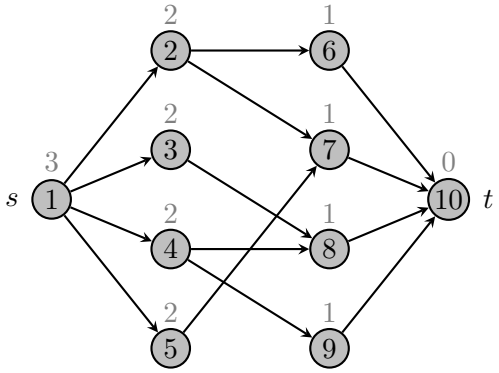
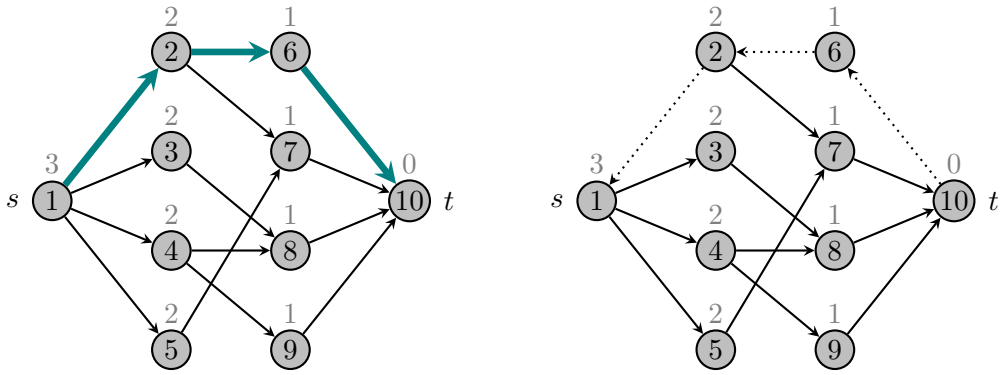


Figura 3.5: Ejemplo de PFM. Red $R^{\boldsymbol{f}}$ en la iteración 1, donde $\boldsymbol{f} = \mathbf{0}$.

Inicialización: Determinamos las distancias $\boldsymbol{d}^t$.

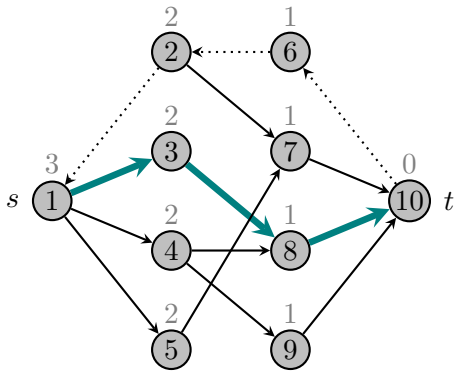
Iteración 1: En la red $R^{\boldsymbol{f}}$, partiendo del nodo fuente s, a través de arcos admisibles encontramos el camino $C^{\boldsymbol{f}} = 1 \hookrightarrow 2 \hookrightarrow 6 \hookrightarrow 10$, por el que enviamos una unidad de flujo.



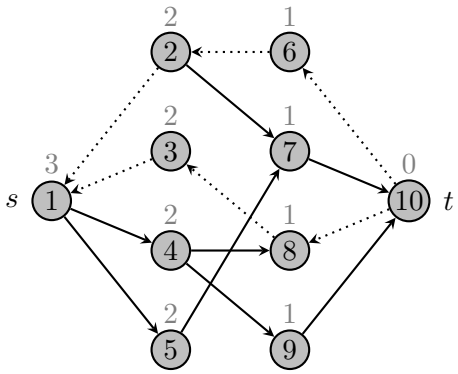
(a) Trayectoria aumentada más corta.

(b) Red $R^{\boldsymbol{f}}$ en la iteración 2.

ITERACIÓN 2: Una vez actualizada la red, buscamos de nuevo un camino dirigido. Partiendo del nodo fuente s, a través de arcos admisibles encontramos el camino $C^{\boldsymbol{f}} = 1 \hookrightarrow 3 \hookrightarrow 8 \hookrightarrow 10$, por el que enviamos una unidad de flujo.

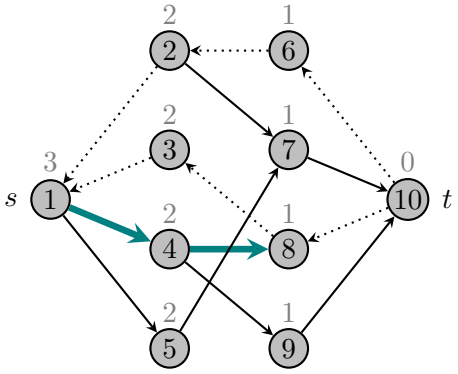


(c) Trayectoria aumentada más corta.

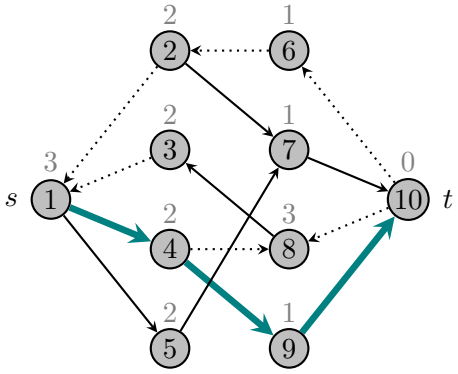


(d) Red $R^{\boldsymbol{f}}$ en la iteración 3.

ITERACIÓN 3: Una vez actualizada la red, buscamos de nuevo un camino dirigido. Partiendo del nodo fuente s, a través de arcos admisibles encontramos el camino $1 \hookrightarrow 4 \hookrightarrow 8$. Como del nodo 8 no sale ningún arco admisible, actualizamos su distancia haciendo $d_8 = \text{mín}\{d_j + 1 : (8, j) \in A'\} = d_3 + 1 = 3$ y volvemos al nodo 4 ($4 = pred_8$) para seguir buscando caminos admisibles. Partiendo del nodo 4, a través de arcos admisibles encontramos el camino $C^{\boldsymbol{f}} = 1 \hookrightarrow 4 \hookrightarrow 9 \hookrightarrow 10$, por el que enviamos una unidad de flujo.

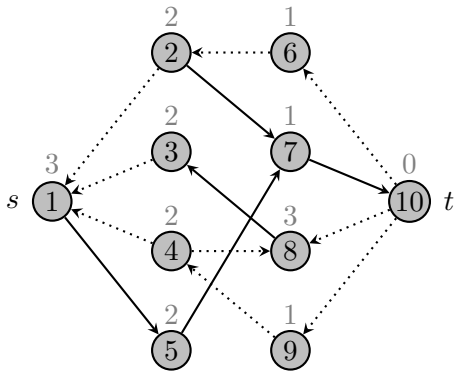


(e) No podemos avanzar en el nodo 8.

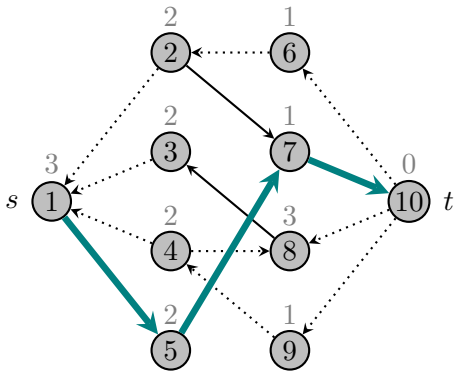


(f) Actualizamos $d_8 = 3$ y buscamos otro camino.

ITERACIÓN 4: Una vez actualizada la red, buscamos de nuevo un camino dirigido. Partiendo del nodo fuente s, a través de arcos admisibles encontramos el camino $C^{\boldsymbol{f}} = 1 \hookrightarrow 5 \hookrightarrow 7 \hookrightarrow 10$, por el que enviamos una unidad de flujo.

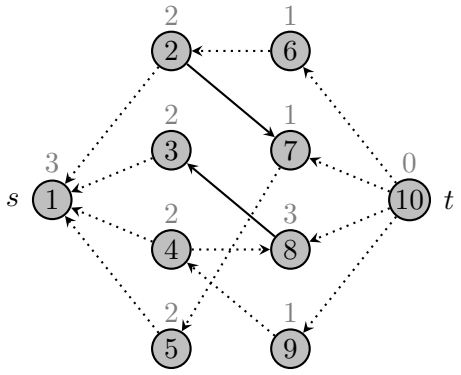


(g) Red R^f en la iteración 4.

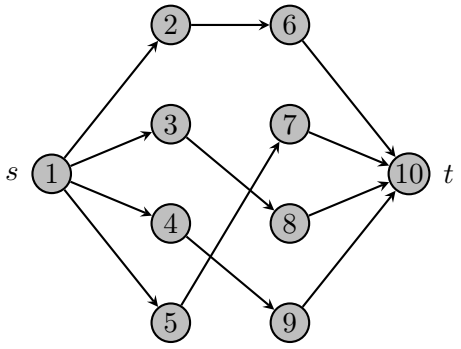


(h) Trayectoria aumentada más corta.

ITERACIÓN 5: En la última iteración del algoritmo, una vez actualizada la red R^f comprobamos que $A_s = \emptyset$, en particular no hay ningún camino admisible desde s hasta t, por lo que el algoritmo termina. El flujo máximo enviado a través de la red original R sigue el siguiente esquema:



(i) Red R^f en la iteración 5.



(j) Distribución del flujo máximo, $F = 4$.

3.2.3. Algoritmo del flujo máximo para capacidades unitarias

Teniendo en cuenta todo lo mencionado en las secciones anteriores podemos presentar formalmente uno de los algoritmos que usaremos para resolver el problema de emparejamiento bipartito de máxima cardinalidad.

Datos: Partimos de un flujo factible $\boldsymbol{f} = \mathbf{0}$ y una red sin costes $R = ((N, A), \boldsymbol{u})$.
mientras $d_s < \text{mín}\{\lceil 2n^{2/3} \rceil, \lceil m^{1/2} \rceil\}$ **hacer**
  aplicar Algoritmo de trayectorias aumentadas más cortas
fin
aplicar Algoritmo de etiquetado (con los datos, $\boldsymbol{f}$ y $R^{\boldsymbol{f}}$, correspondientes a la última iteración del algoritmo de trayectorias aumentadas más cortas)

Algoritmo 6: Algoritmo del flujo máximo para capacidades unitarias.

Este algoritmo termina, devolviendo un flujo máximo sobre R, cuando cualquiera de los dos algoritmos de los que es combinación termina.

Capítulo 4

El problema de emparejamiento bipartito

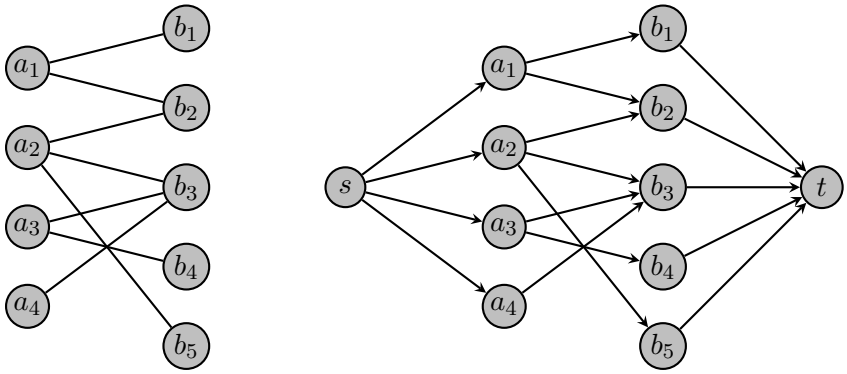
El problema de emparejamiento bipartito es un caso particular del problema de emparejamiento en el cual los objetos están divididos en dos grupos y deseamos emparejarlos intentando cumplir algún tipo de objetivo. Consideraremos dos versiones de este problema: el problema bipartito de máxima cardinalidad (Sección 4.1), que consiste en encontrar un emparejamiento máximo y el problema bipartito ponderado (Sección 4.2), donde hay un coste asociado a cada arco y se desea encontrar el emparejamiento que empareje todos los nodos de la red con menor coste. Nótese que el problema de máxima cardinalidad se puede ver como un problema ponderado donde los costes de todos los arcos son iguales a -1, por tanto, los algoritmos que presentemos para resolver el problema ponderado pueden usarse también para resolver el de máxima cardinalidad. Sin embargo, encontraremos algoritmos específicos para el problema de máxima cardinalidad que lo resolverán más eficientemente.

Veremos que los problemas bipartitos son fáciles de resolver, ya que los podemos modelar como casos particulares de los problemas de flujo en redes vistos en los anteriores capítulos.

4.1. El problema bipartito de máxima cardinalidad

Como explicamos previamente, en el problema bipartito de máxima cardinalidad se busca un emparejamiento que asocie el mayor número de nodos posibles de un grafo bipartito no dirigido $G = (N_1 \cup N_2, A)$. Una manera de resolver este problema es transformarlo en un problema de flujo máximo haciendo modificaciones sobre el grafo G como se describe a continuación.

Para empezar convertiremos el grafo no dirigido G en uno dirigido, haciendo que cada arco tenga origen en un nodo de N_1 y termine en un nodo de N_2. Además introduciremos dos nuevos nodos, un nodo fuente s y un nodo sumidero t, y nuevos arcos conectando s con cada nodo de N_1 y conectando cada nodo de N_2 con t. Obtenemos así el grafo $G' = (N', A')$, a partir del cual construiremos la red R' fijando la capacidad de cada arco igual a 1. Tras estos sencillos pasos obtenemos $R' = ((N', A'), \boldsymbol{u})$ que además es una red simple.



(a) Grafo original G. (b) Red R', $u_{ij} = 1\ \forall (i,j) \in A'$.

Figura 4.1: Transformando un problema bipartito de máxima cardinalidad en un problema de flujo máximo con capacidades unitarias.

Veamos cómo se establece una correspondencia entre un emparejamiento de cardinal k en el grafo original G y un flujo entero de valor k en la red R'.

Dado un emparejamiento $\{(a_{i_1}, b_{i_1}), (a_{i_2}, b_{i_2}), \ldots, (a_{i_k}, b_{i_k})\} \subseteq (N_1 \times N_2)^k$ de cardinalidad k sobre G, construiremos el flujo correspondiente en R'. Para satisfacer las restricciones de balance de flujo fijamos el flujo de cada arco $(s, a_{i_j}), (a_{i_j}, b_{i_j})$ y (b_{i_j}, t) igual a 1 para todo $j = 1, 2, \ldots, k$, obteniendo así un flujo de valor k desde la fuente al sumidero.

De manera análoga, dado un flujo entero de valor k del nodo fuente al nodo sumidero en la red R', obtendremos el emparejamiento correspondiente en el grafo original G. El flujo de valor k se descompone en k caminos de la forma $s \hookrightarrow a_{i_j} \hookrightarrow b_{i_j} \hookrightarrow t$, con $j = 1, 2, \ldots, k$. Dado que todas los capacidades en R' son unitarias tenemos asegurado que ningún nodo forma parte de más de un camino, por tanto, los arcos $\{(a_{i_1}, b_{i_1}), (a_{i_2}, b_{i_2}), \ldots, (a_{i_k}, b_{i_k})\}$ definen un emparejamiento de cardinalidad k sobre G.

Establecida por tanto la correspondencia entre los emparejamientos en G y los flujos en R' concluimos que, para resolver un problema de emparejamiento bipartito de máxima

cardinalidad podemos resolver el problema de flujo máximo correspondiente con el algoritmo de trayectorias aumentadas visto en la Sección 3.1, que proporciona un flujo entero óptimo en tiempo $O(mF)$. Además, por ser la red R' una red con capacidades unitarias, este tiempo de ejecución es mejorable con el algoritmo visto en la Sección 3.2, con el que conseguimos un tiempo de computación de $O(\sqrt{n}m)$.

4.1.1. Algoritmo para el emparejamiento bipartito de máxima cardinalidad

A continuación presentamos un algoritmo específico para resolver el problema de emparejamiento bipartito de máxima cardinalidad, sin necesidad de transformar el grafo original, que se apoya en las propiedades de la diferencia simétrica y en el algoritmo de búsqueda de caminos aumentadores (Sección 2.3.1).

Partiendo de un emparejamiento factible, por ejemplo $M = \emptyset$, el algoritmo busca caminos aumentadores que comiencen en los nodos no emparejados del grafo, $p \in N$. Si se encuentra tal camino P entonces se reemplaza M por $M \oplus P$, obteniendo un emparejamiento de mayor cardinalidad (véase la Propiedad 2.3). En caso de no existir este camino se elimina del grafo el nodo p y todos los arcos incidentes en él, pues por el Teorema 2.6 sabemos que existe un emparejamiento máximo en el cual p no está emparejado.

Usando el Teorema del camino aumentador es fácil ver que este algoritmo encuentra un emparejamiento óptimo, pues en cada iteración reduce el número de nodos no emparejados en al menos uno, ya sea emparejando algún nodo o eliminándolo del grafo. Dado que los nodos que están emparejados inicialmente permanecen emparejados a lo largo de la ejecución del algoritmo (por la Propiedad 2.3), cuando el algoritmo termina cada nodo en el subgrafo obtenido está emparejado. Por tanto, M debe ser un emparejamiento máximo sobre dicho subgrafo que, por el Teorema 2.6, también es un emparejamiento máximo sobre G.

Véase que hemos reducido el problema de emparejamiento de máxima cardinalidad a encontrar si el grafo contiene o no un camino aumentador que empiece en un nodo p.

Este algoritmo, que siempre establece un emparejamiento de máxima cardinalidad cuando se aplica a grafos bipartitos, tiene un tiempo de $O(nm)$. Esto es debido a que el algoritmo ejecuta un máximo de n veces las subrutinas de búsqueda y aumentar. Mientras que el proceso de aumentar requiere claramente un tiempo de $O(n)$, ya comprobamos en el segundo capítulo que el proceso de búsqueda requiere un tiempo de $O(m)$ por cada ejecución.

Datos: Partimos del emparejamiento factible $M = \emptyset$ sobre un grafo bipartito no dirigido $G = (N_1 \cup N_2, A)$.
para cada *nodo* $p \in N$ **hacer**
  si *el nodo p no está emparejado* **entonces**
    subrutina búsqueda(p, found)
    si *found* = *true* **entonces**
      subrutina aumentar
    en otro caso
      borrar el nodo p y todos los arcos incidentes a él en G
    fin
  fin
fin

---

subrutina aumentar
identificar el camino aumentador P empezando en q y usando los predecesores
actualizar el emparejamiento haciendo $M \leftarrow M \oplus P$

Algoritmo 7: Algoritmo del emparejamiento bipartito de máxima cardinalidad.

Ejemplo de resolución

Veamos un ejemplo de resolución de un problema de emparejamiento bipartito de máxima cardinalidad usando el algoritmo presentado. Partimos del siguiente grafo G:

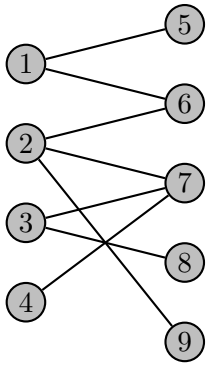
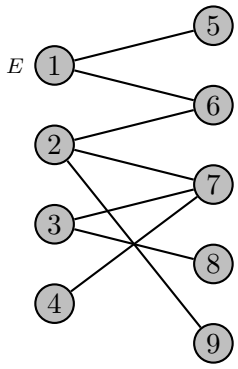


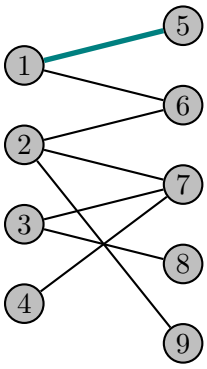
Figura 4.2: Ejemplo de problema de emparejamiento bipartito de máxima cardinalidad.

INICIALIZACIÓN: Partimos del emparejamiento nulo $M_0 = \emptyset$ sobre el grafo bipartito G.

ITERACIÓN 1: Tomemos el nodo $1 \in N$ que no está emparejado en M_0. Ejecutamos el algoritmo búsqueda(1, found), que asigna etiqueta Par al nodo 1 y examina su lista de adyacencia A_1. Encontramos el camino aumentador $P_1 = 1 \hookrightarrow 5$, con lo que found = *true* y conseguimos un emparejamiento de cardinalidad 1, $M_1 = M_0 \oplus P_1$.

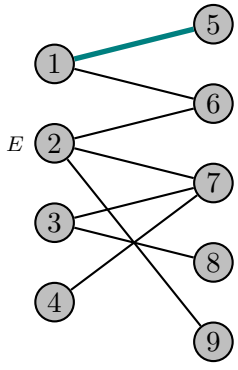


(a) Etiquetas asignadas en la iteración 1.

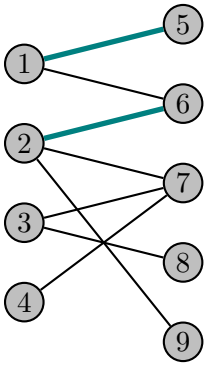


(b) $M_1 = \{(1, 5)\}$.

ITERACIÓN 2: Tomemos el nodo $2 \in N$ que no está emparejado en M_1. Ejecutamos el algoritmo búsqueda(2, found), que asigna etiqueta Par al nodo 2 y examina su lista de adyacencia A_2. Encontramos el camino aumentador $P_2 = 2 \hookrightarrow 6$, con lo que found = *true* y conseguimos un emparejamiento de cardinalidad 2, $M_2 = M_1 \oplus P_2$.



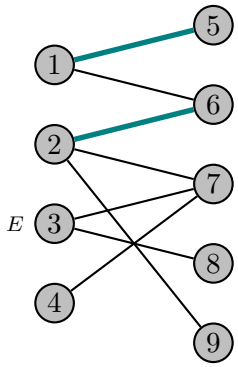
(c) Etiquetas asignadas en la iteración 2.



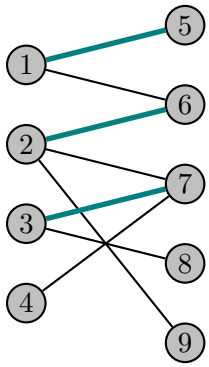
(d) $M_2 = \{(1, 5), (2, 6)\}$.

ITERACIÓN 3: Tomemos el nodo $3 \in N$ que no está emparejado en M_2. Ejecutamos el algoritmo búsqueda(3, found), que asigna etiqueta Par al nodo 3 y examina su lista de adyacencia A_3. Encontramos el camino aumentador $P_3 = 3 \hookrightarrow 7$, con lo que found = *true*

y conseguimos un emparejamiento de cardinalidad 3, $M_3 = M_2 \oplus P_3$.

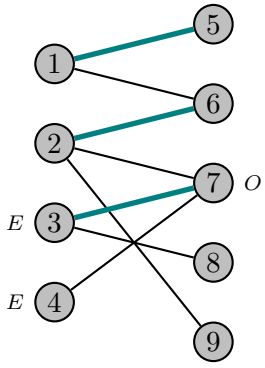


(e) Etiquetas asignadas en la iteración 3.

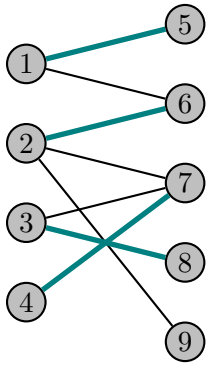


(f) $M_3 = \{(1,5),(2,6),(3,7)\}$.

ITERACIÓN 4: Tomemos el nodo $4 \in N$ que no está emparejado en M_3. Ejecutamos el algoritmo búsqueda(4, found), que asigna etiqueta Par al nodo 4 y examina su lista de adyacencia. Como el nodo $7 \in A_4$ ya está emparejado, se le asigna etiqueta Impar y se añande a $\mathcal{L}$. Examinamos ahora el nodo 7, que está emparejado al nodo 3, con lo cual se le asigna etiqueta Par al nodo 3 y se añade a $\mathcal{L}$. Al examinar la lista de adyacencia A_3 encontramos el camino aumentador $P_4 = 4 \hookrightarrow 7 \hookrightarrow 3 \hookrightarrow 8$, con lo que found $=$ *true* y conseguimos un emparejamiento de cardinalidad 4, $M_4 = M_3 \oplus P_4$.



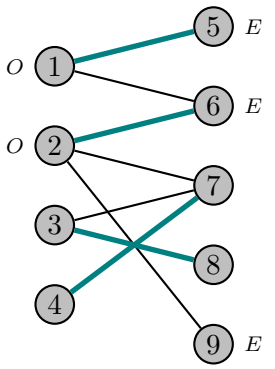
(g) Etiquetas asignadas en la iteración 4.



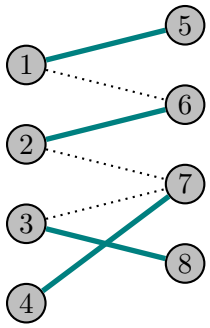
(h) $M_4 = \{(1,5),(2,6),(3,8),(4,7)\}$.

ITERACIÓN 5: Tomemos el nodo $9 \in N$ que no está emparejado en M_4. Ejecutamos el algoritmo búsqueda(9, found), que asigna etiqueta Par al nodo 9 y examina su lista de adyacencia. Como el nodo $2 \in A_9$ ya está emparejado, se le asigna etiqueta Impar y se

añande a $\mathcal{L}$. Examinamos ahora el nodo 2, que está emparejado al nodo 6, con lo cual se le asigna etiqueta Par al nodo 6 y se añade a $\mathcal{L}$. Al examinar la lista de adyacencia A_6 encontramos el nodo 1, que ya está emparejado, por lo que se le asigna etiqueta Impar y se añade a $\mathcal{L}$. Examinamos ahora el nodo 1, que está emparejado al nodo 5, con lo cual se le asigna etiqueta Par al nodo 5 y se añade a $\mathcal{L}$. Después de examinar la lista de adyacencia A_5, nos encontramos con una lista $\mathcal{L} = \emptyset$ y volvemos al algoritmo principal. Como found = *false* procedemos a eliminar el nodo 9 y todos los arcos incidentes en él, obteniendo el emparejamiento M_4 sobre el subgrafo final.



(i) Etiquetas asignadas en la iteración 5.



(j) Emparejamiento M_4 sobre el subgrafo.

Ahora bien, como en el subgrafo todos los nodos están emparejados el algoritmo termina, proporcionando un emparejamiento de máxima cardinalidad sobre el grafo bipartito original.

4.2. El problema bipartito ponderado

Este es el único caso de las cuatro versiones del problema de emparejamiento que se estudia en la asignatura de Programación Lineal y Entera del Grado. El problema de emparejamiento bipartito ponderado también se conoce como problema de asignación por lo que, a partir de ahora, nos referiremos a él indistintamente de cualquiera de las dos maneras. El planteamiento de este problema y la información correspondiente al método húngaro está basado en los apuntes de González-Díaz (2018), donde se habla siempre de problema de asignación. Las cuestiones que presentamos a mayores de lo estudiado en el Grado están basadas en el Capítulo 12 de Ahuja et al. (1993).

Dada una red bipartita $R = ((N_1 \cup N_2, A), \boldsymbol{u}, \boldsymbol{c})$ con $u_{ij} = 1$ para todo $(i, j) \in A$, el problema de asignación consiste en emparejar cada elemento de N_1 con un elemento de N_2

a coste mínimo. Suponemos que $G = (N_1 \cup N_2, A)$ es un grafo dirigido, si no lo fuera lo convertimos en un grafo dirigido haciendo que cada arco vaya de un nodo de N_1 a uno de N_2. También podemos suponer sin pérdida de generalidad que $|N_1| = |N_2| = n_1$, si no fuera así siempre podemos añadir nodos artificiales para representar elementos no asignados.

Recordemos que el problema de asignación es un caso particular del PFCM, por lo que se puede escribir como un problema de programación lineal como sigue:

$$\begin{aligned}
\text{Minimizar} \quad & \sum_{(i,j)\in A} c_{ij} f_{ij} \\
\text{Sujeto a} \quad & \sum_{\{j:(i,j)\in A\}} f_{ij} = 1, \quad \text{para todo } i \in N_1 \\
& \sum_{\{j:(j,i)\in A\}} f_{ji} = 1, \quad \text{para todo } i \in N_2 \\
& f_{ij} \geq 0, \quad \text{para todo } (i,j) \in A.
\end{aligned}$$

Dado que podemos formular el problema de emparejamiento bipartito ponderado como este tipo especial de problema de flujo en redes, no es de extrañar que la mayoría de los algoritmos que usamos para resolverlo sean adaptaciones de algoritmos para resolver PFCMs. Además, la estructura especial de este tipo de problemas nos permite simplificar los algoritmos, de manera que al aplicarlos al problema de asignación sean más eficientes computacionalmente.

A continuación presentamos un par de algoritmos para la resolución del problema de emparejamiento bipartito ponderado.

4.2.1. Algoritmo de caminos más cortos sucesivos

Consiste en implementar directamente el algoritmo de caminos más cortos sucesivos para PFCMs descrito en la Sección 2.4. Teniendo en cuenta que cuando se aplica este algoritmo a un problema de emparejamiento bipartito ponderado aumentará una unidad de flujo en cada iteración, sabemos que tardará $n_1 = |N_1| = |N_2|$ iteraciones en encontrar el emparejamiento óptimo. Por tanto, si denotamos por $S(n, m, C)$ al tiempo necesario para resolver el problema del camino más corto sin arcos de longitud negativa (donde C es el coste más alto de la red), el algoritmo de caminos más cortos sucesivos encuentra un emparejamiento óptimo en un tiempo de $O(n_1 S(n, m, C))$.

4.2.2. El método húngaro

El método húngaro es uno de los algoritmos más antiguos para resolver el problema de asignación (H. Kuhn, 1955). Además es uno de esos algoritmos que explota las relaciones entre las formulaciones primal y dual del problema para resolverlo. Partimos del supuesto de que todas las asignaciones son posibles, es decir $A = N_1 \times N_2$, si esto no fuera cierto podemos añadir al problema inicial los arcos que faltan con costes muy altos. Nótese que si alguno de estos arcos añadidos forma parte de nuestra solución óptima, esto quiere decir que el problema inicial no tiene solución factible. Se sabe que el método húngaro, que es un algoritmo polinomial, encuentra una solución óptima al problema de asignación en un tiempo, en el peor caso, de $O(n_1^3)$.

Presentaremos los costes del problema de asignación como una matriz $\boldsymbol{C} = (c_{ij})$ de tamaño $n_1 \times n_1$, donde denotamos por w_i al mínimo de la fila i. Construimos la matriz $\boldsymbol{C'}$ restando a cada fila el mínimo de esa fila, es decir, $c'_{ij} = c_{ij} - w_i$. En esta nueva matriz llamamos v_j al mínimo de la columna j y, a partir de ella construimos la matriz $\hat{\boldsymbol{C}}$, llamada **matriz reducida**, restando a cada columna el mínimo de esa columna, es decir,

$$\hat{c}_{ij} = c'_{ij} - v_j = c_{ij} - w_i - v_j.$$

Esta última matriz tendrá, por construcción, al menos un cero en cada fila y en cada columna. Por el Teorema de las holguras complementarias (1.8) sabemos que si encontramos un flujo factible $\boldsymbol{f}$, tal que los flujos con valor 1 se corresponden con celdas en las que $\hat{c}_{ij} = 0$, tendríamos un óptimo. Este flujo factible $\boldsymbol{f}$ tendrá n_1 aristas con valor 1 y el resto con valor 0, lo cual se corresponde con elegir n_1 celdas de la matriz reducida (una celda de cada fila y una de cada columna).

Veamos un primer ejemplo de un problema de asignación, con $|N_1| = |N_2| = 5$ y costes dados por $\boldsymbol{C}$, donde encontramos el flujo factible $\boldsymbol{f}$ en la matriz reducida $\hat{\boldsymbol{C}}$:

c_{ij}	*1*	*2*	*3*	*4*	*5*
1	7	2	**1**	9	4
2	9	6	9	**5**	5
3	3	8	3	**1**	8
4	7	9	4	**2**	2
5	8	**4**	7	4	8

c'_{ij}	*1*	*2*	*3*	*4*	*5*
1	6	1	**0**	8	3
2	4	1	4	**0**	**0**
3	**2**	7	2	0	7
4	5	7	2	0	0
5	4	**0**	3	0	4

$\hat{c}_{ij}$	*1*	*2*	*3*	*4*	*5*
1	4	1	**0**	8	3
2	2	1	4	**0**	0
3	**0**	7	2	0	7
4	3	7	2	0	**0**
5	2	**0**	3	0	4

Figura 4.3: Problema de asignación resuelto con el método húngaro en una etapa. En la primera matriz, $\boldsymbol{C}$, señalamos los w_i y en la segunda, $\boldsymbol{C'}$, los v_j.

En este ejemplo encontramos el flujo factible, y óptimo, en la matriz reducida. Eligiendo únicamente celdas con el valor 0 tenemos una solución óptima:

$$f_{13} = 1, f_{24} = 1, f_{31} = 1, f_{45} = 1, f_{52} = 1, \text{ con coste } c = 1 + 5 + 3 + 2 + 4 = 15.$$

Es fácil ver que existe otra solución óptima con el mismo coste, tomando:

$$f_{13} = 1, f_{25} = 1, f_{31} = 1, f_{44} = 1, f_{52} = 1.$$

Estas dos soluciones óptimas se corresponden con los siguientes emparejamientos sobre R:

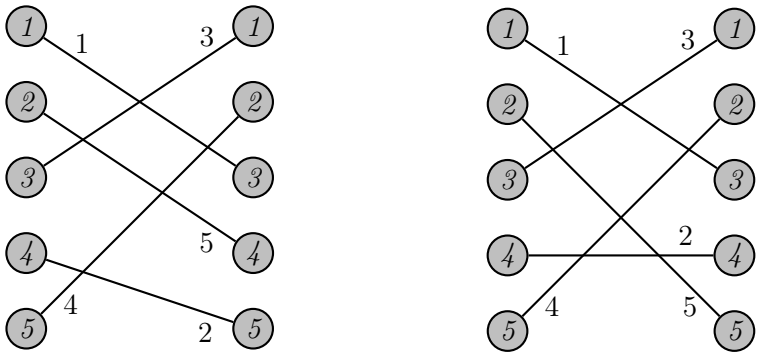


Figura 4.4: Soluciones obtenidas con el método húngaro.

Sin embargo, si no encontramos un flujo factible en la matriz reducida que conste únicamente de celdas con ceros, tendremos que desarrollar un poco más el método para poder encontrar una solución óptima del problema de emparejamiento bipartito ponderado.

Una vez construida la matriz de costes reducida en la cual no encontramos un flujo factible $\boldsymbol{f}$, buscamos el mínimo número de filas y columnas de $\hat{\boldsymbol{C}}$ que es necesario tachar para cubrir todos los ceros. Mientras este número, al que denotaremos $\mathsf{cubrir}(\hat{\boldsymbol{C}})$, sea menor que n_1 actualizamos la matriz reducida del siguiente modo: siendo Δ el mínimo de los elementos no cubiertos de $\hat{\boldsymbol{C}}$, restamos Δ a los elementos de $\hat{\boldsymbol{C}}$ que no se hayan cubierto y sumamos Δ a los elementos de $\hat{\boldsymbol{C}}$ que se hayan cubierto dos veces. Sobre esta nueva matriz volvemos a calcular $\mathsf{cubrir}(\hat{\boldsymbol{C}})$ hasta que sea igual a n_1, en cuyo caso podemos encontrar un emparejamiento óptimo sobre la matriz $\hat{\boldsymbol{C}}$, la cual seguimos actualizando con este procedimiento.

Observación. Aunque el proceso de buscar el número mínimo de filas y columnas que hay que tachar para cubrir todos los ceros de la matriz reducida puede parecer fácil a simple vista, no lo es tanto para problemas relativamente grandes por lo que su implementación no es inmediata. Sin embargo, se puede demostrar que este paso es equivalente a resolver

un problema de flujo máximo, para el cual conocemos algoritmos polinomiales.

Datos: Partimos de la red bipartita R con capacidades unitarias y matriz de costes $\boldsymbol{C}$.
calcular la matriz reducida $\hat{\boldsymbol{C}}$ dada por $\hat{c}_{ij} = c_{ij} - w_i - v_j$
calcular $\mathsf{cubrir}(\hat{\boldsymbol{C}})$ para la primera matriz reducida
mientras *cubrir*$(\hat{\boldsymbol{C}}) < n_1$ **hacer**
  $\Delta \leftarrow \text{mín}_{\text{no tachados}}\{\hat{c}_{ij}\}$
  $\hat{c}_{ij} \leftarrow \hat{c}_{ij} - \Delta$, si $\hat{c}_{ij}$ no está tachado
  $\hat{c}_{ij} \leftarrow \hat{c}_{ij} + \Delta$, si $\hat{c}_{ij}$ está tachado dos veces
  actualizar $\mathsf{cubrir}(\hat{C})$
fin

Algoritmo 8: Método húngaro.

Ejemplo de resolución

Veamos un ejemplo de problema de asignación resuelto con el método húngaro, que ilustre el algoritmo por completo.

c_{ij}	*1*	*2*	*3*	*4*	*5*
1	2	3	5	1	4
2	-1	1	3	6	2
3	-2	4	3	5	0
4	1	3	4	1	4
5	7	1	2	1	2

(a) Problema de asignación

$\hat{c}_{ij}$	*1*	*2*	*3*	*4*	*5*
1	1	2	3	0	2
2	0	2	3	7	2
3	0	6	4	7	**1**
4	0	2	2	0	2
5	6	0	0	0	0

(b) Primera matriz reducida

	1	*2*	*3*	*4*	*5*
1	1	**1**	2	0	1
2	0	1	2	7	1
3	0	5	3	7	0
4	0	1	1	0	1
5	7	0	0	1	0

(c) Segunda matriz

	1	*2*	*3*	*4*	*5*
1	1	0	1	0	1
2	0	0	1	7	1
3	0	4	2	7	0
4	0	0	0	0	1
5	8	0	0	2	1

(d) Tercera matriz

Figura 4.5: Problema de asignación resuelto con el método húngaro.

En la primera matriz vemos representados los costes c_{ij} de los arcos del problema de

asignación original. Tras construir la matriz reducida tachamos la última fila y las columnas *1* y *4* para cubrir todos los ceros, como $3 < 5 = n_1$ actualizamos la matriz reducida. En este caso $\Delta = 1$, con lo que construimos la segunda matriz. Ahora tachamos la última fila y las columnas *1*, *4* y *5* para cubrir todos los ceros, como $4 < 5 = n_1$ actualizamos de nuevo la matriz reducida. Volvemos a obtener $\Delta = 1$, con lo que construimos la tercera matriz. En esta ya encontramos varios flujos factibles que se corresponden con emparejamientos óptimos, veamos cuáles son:

$$\begin{cases} f_{12} = 1, f_{21} = 1, f_{35} = 1, f_{44} = 1, f_{53} = 1, \text{ con coste } c = 3 - 1 + 0 + 1 + 2 = 5 \\ f_{14} = 1, f_{21} = 1, f_{35} = 1, f_{42} = 1, f_{53} = 1, \text{ con coste } c = 1 - 1 + 0 + 3 + 2 = 5 \\ f_{14} = 1, f_{21} = 1, f_{35} = 1, f_{43} = 1, f_{52} = 1, \text{ con coste } c = 1 - 2 + 0 + 4 + 1 = 5 \\ f_{14} = 1, f_{22} = 1, f_{35} = 1, f_{41} = 1, f_{53} = 1, \text{ con coste } c = 1 + 1 + 0 + 1 + 2 = 5 \end{cases}$$

Que se representan como:

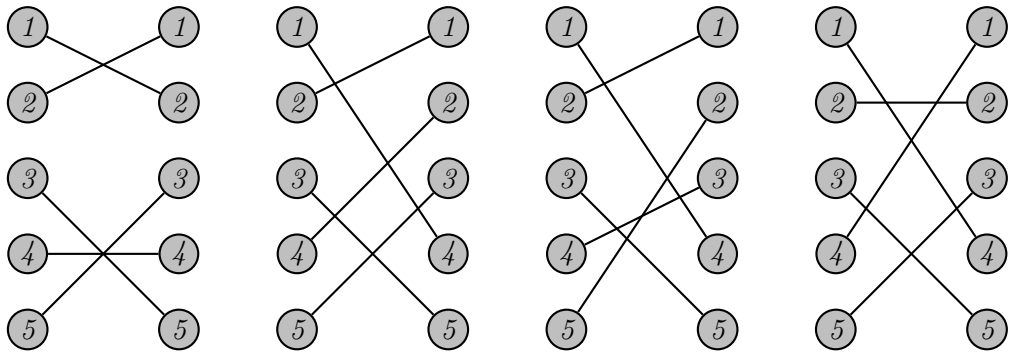


Figura 4.6: Emparejamientos óptimos obtenidos mediante el método húngaro.

Capítulo 5

El problema de emparejamiento no bipartito

El problema de emparejamiento no bipartito es un caso más general que el emparejamiento bipartito estudiado en el capítulo anterior, ya que los objetos que deseamos emparejar no tienen por qué estar divididos en dos grupos (el grafo subyacente a la red no tiene por qué ser bipartito). Esta clase de problemas es más difícil de resolver que los bipartitos, ya que no se pueden reducir a un problema estándar de flujo en redes, por eso requieren de algoritmos especializados.

5.1. El problema no bipartito de máxima cardinalidad

El problema de emparejamiento no bipartito de máxima cardinalidad consiste en encontrar un emparejamiento que asocie el mayor número de nodos de un grafo no bipartito y no dirigido $G = (N, A)$.

Dificultades con el algoritmo para el emparejamiento bipartito de máxima cardinalidad

En esta sección comentamos los inconvenientes que presenta el algoritmo de búsqueda de caminos aumentadores que presentamos en la Sección 2.3.1 cuando lo aplicamos a grafos no bipartitos y, por tanto, las dificultades que presenta el algoritmo para el problema de emparejamiento bipartito de máxima cardinalidad descrito en la Sección 4.1.1.

¿Podemos asegurar que este algoritmo de búsqueda trabaja correctamente? Está claro que si el algoritmo encuentra un camino aumentador sobre el grafo al que se aplica significa que este existe pero, ¿podemos concluir que no exista tal camino aumentador cuando el

algoritmo no lo encuentra? Probaremos que esto último se verifica si el grafo posee la propiedad de la etiqueta única (que definimos a continuación); en otro caso, la conclusión será incorrecta.

Se dice que un grafo posee la **propiedad de la etiqueta única** con respecto a un emparejamiento dado M y un nodo raíz p, si el algoritmo búsqueda(p, found) descrito en la Sección 2.3.1 asigna la misma etiqueta (Par o Impar) a cada nodo marcado independientemente del orden en el que los examine.

Propiedad 5.1. *Si un grafo posee la propiedad de la etiqueta única, entonces el algoritmo de búsqueda siempre encontrará un camino aumentador en caso de que exista.*

Demostración. Supongamos que el grafo contiene un camino aumentador $p \hookrightarrow i_1 \hookrightarrow j_1 \hookrightarrow i_2 \hookrightarrow j_2 \hookrightarrow \ldots \hookrightarrow i_j \hookrightarrow j_l \hookrightarrow q$ desde el nodo raíz p hasta q con respecto al emparejamiento M. Si se examinan los nodos en este orden, el algoritmo asignará la etiqueta Par a los nodos $p, j_1, j_2, \ldots, j_l$, mientras que los nodos $i_1, i_2, \ldots, i_l, q$ tendrán etiqueta Impar. Como por hipótesis el grafo posee la propiedad de la etiqueta única, al algoritmo asignará estas mismas etiquetas sea cual el orden en el que examina los nodos. Así, el procedimiento de búsqueda siempre asignará etiqueta Impar al nodo q y por tanto encontrará un camino aumentador. □

Los grafos bipartitos satisfacen trivialmente la propiedad de la etiqueta única, con respecto a cualquier emparejamiento M y cualquier nodo raíz p. Es por esto que, teniendo en cuenta la propiedad anterior, el algoritmo para el emparejamiento bipartito de máxima cardinalidad siempre encontrará un emparejamiento óptimo. Veamos a continuación por qué este algoritmo falla cuando se aplica a una red no bipartita.

Un grafo no bipartito puede no satisfacer la propiedad de la etiqueta única (véase la Propiedad 2.8), caso en el cual el algoritmo de búsqueda de un camino aumentador puede fallar aunque este camino exista. Consideremos la situación dada en la siguiente figura:

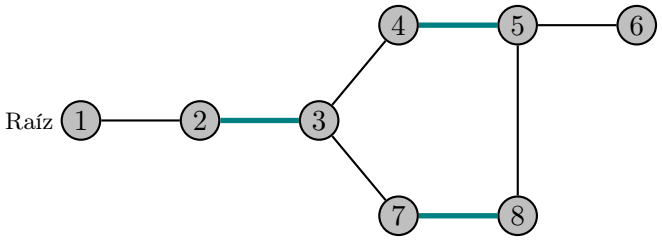


Figura 5.1: Problema de emparejamiento no bipartito.

Si el nodo 5 recibe su etiqueta a través del camino $1 \hookrightarrow 2 \hookrightarrow 3 \hookrightarrow 4 \hookrightarrow 5$ recibirá la etiqueta Par. En este caso, al examinar la lista de adyacencia A_5, se le atribuye al nodo 6 la etiqueta Impar descubriendo así un camino aumentador sobre el grafo de partida. Sin embargo, si el nodo 5 recibe etiqueta Impar al ser alcanzado a través del camino $1 \hookrightarrow 2 \hookrightarrow 3 \hookrightarrow 7 \hookrightarrow 8 \hookrightarrow 5$, sólamente examinamos el arco emparejado que sale de él. Con esto llegamos al nodo 4 sin haber examinado el arco no emparejado $(5, 6)$, por lo tanto, en este caso el algoritmo presentado en la Sección 2.3.1 no encuentra ningún camino aumentador.

5.1.1. Algoritmo para el emparejamiento no bipartito de máxima cardinalidad

El algoritmo que presentamos en esta sección es análogo al usado para el caso bipartito, con la excepción de que modifica el proceso de búsqueda del camino aumentador. El proceso de búsquedaNB(p,found) difiere del presentado en la Sección 2.3.1 en que se utiliza una subrutina adicional, llamada contraer(i,j), que convierte una corola en un pseudo-nodo y actualiza las estructuras de datos tal y como se explicó en la Sección 2.2.

Expliquemos cómo funciona el algoritmo para el problema de emparejamiento no bipartito de máxima cardinalidad. Cuando estamos buscando un camino aumentador desde un nodo no emparejado $p \in N$ y nos encontramos con un nodo, digamos i, al que se le puede asignar una etiqueta distinta a la que ya tiene, el algoritmo suspende el proceso de búsqueda. En este momento hemos encontrado dos caminos alternados con distinta paridad que llegan al nodo i desde p. Si examinamos los predecesores de esos caminos, llegará un momento en el que encontremos un nodo común a ambos. Así, los arcos examinados hasta ese momento constituyen una corola y el primer nodo común a ambos caminos alternados, que tendrá etiqueta Par, es la base de la corola.

Llegados a este punto no hay más que contraer la corola en un pseudo-nodo, actualizar los datos y continuar con el proceso de búsqueda. Nótese que podríamos necesitar contraer varias corolas distintas antes de encontrar un camino aumentador en el grafo contraído o antes de quedarnos sin nodos que examinar, lo cual indica que el grafo no contiene caminos aumentadores que comiencen en el nodo raíz p. Cuando encontramos un camino aumentador del nodo p a algún otro nodo no emparejado q debemos comprobar si este camino contiene algún pseudo-nodo. Si es así expandimos las corolas representadas por estos pseudo-nodos hasta que el camino aumentador no contenga pseudo-nodos.

Cada vez que se ejecuta la subrutina contraer(i,j) se crea un nuevo pseudo-nodo en el grafo. Para hacer un seguimiento de estos nodos creados a lo largo del proceso de búsque-

da los numeraremos como $n+1, n+2, \ldots$ Por lo que i será un pseudo-nodo si, y solo si, $i > n$.

Datos: Partimos de un emparejamiento factible $M = \emptyset$ sobre un grafo no bipartito y no dirigido $G = (N, A)$.
para cada *nodo* $p \in N$ **hacer**
  si *el nodo p no está emparejado* **entonces**
    subrutina búsquedaNB(p, found)
    si *found* = *true* **entonces**
      subrutina aumentar
    en otro caso
      borrar el nodo p y todos los arcos incidentes a él en G
    fin
  fin
fin

---

subrutina aumentar
identificar el camino aumentador P' empezando en q y usando predecesores
si *el camino* P' *contiene pseudo-nodos* **entonces**
  expandir las correspondientes corolas y obtener el camino aumentador P en el grafo original
fin
actualizar el emparejamiento haciendo $M \leftarrow M \oplus P$

Algoritmo 9: Algoritmo del emparejamiento no bipartito de máxima cardinalidad.

A continuación presentamos dos resultados, cuyas demostraciones se pueden consultar en la Sección 12.6 de Ahuja et al. (1993), que nos permitirán asegurar que el algoritmo encuentra un emparejamiento máximo.

Lema 5.2. *Si el grafo contraído G^c contiene un camino aumentador P^c que empieza en el nodo raíz p (o en el pseudo-nodo que contiene a p) con respecto a un emparejamiento M^c, entonces el grafo original G contiene un camino aumentador empezando en el nodo p con respecto al emparejamiento M.*

Este lema prueba que si descubrimos un camino aumentador en el grafo contraído, podemos usar este camino para identificar un camino aumentador en el grafo original. Además cuando contraemos una corola no añadimos ningún camino aumentador más allá de los contenidos en el grafo original. Faltaría probar el resultado opuesto, que cuando contraemos una corola no pasamos por alto ningún camino aumentador de la red original.

Datos: Partimos de un emparejamiento factible M sobre un grafo no bipartito y no dirigido G y un nodo p no emparejado.
fijar $A_i^c \leftarrow A_i$ para todos los nodos $i \in N$
found $\leftarrow$ *false*
desetiquetar todos los nodos
etiquetar el nodo p como Par e inicializar $\mathcal{L} = \{p\}$
mientras $\mathcal{L} \neq \emptyset$ **hacer**
  eliminar un nodo i de $\mathcal{L}$
  si *el nodo i tiene etiqueta Par* **entonces**
    subrutina examinarNB-Par(i,found)
  en otro caso
    subrutina examinarNB-Impar(i,found)
  fin
  si *found* = *true* **entonces**
    return
  fin
fin

---

subrutina examinarNB-Par(i, found)
para *todo nodo* $j \in A_i^c$ **hacer**
  si j *tiene etiqueta Par* **entonces**
    subrutina contraer(i,j)
    return
  fin
  si j *no está emparejado* **entonces**
    fijar $q \leftarrow j$ y $pred_q \leftarrow i$
    found $\leftarrow$ true
    return
  fin
  si j *está emparejado y no etiquetado* **entonces**
    $pred_j \leftarrow i$
    etiquetar j como Impar
    añadir el nodo j a $\mathcal{L}$
  fin
fin

Algoritmo 10: Subrutina búsquedaNB(p,found).

subrutina examinarNB-Impar(i, found)
sea j el nodo emparejado al nodo i
si *j tiene etiqueta Impar* **entonces**
    subrutina contraer(i,j)
    return
fin
si *el nodo j no está emparejado ni etiquetado* **entonces**
    $pred_j \leftarrow i$
    etiquetar el nodo j como Par
    añadir el nodo j a $\mathcal{L}$
fin

---

subrutina contraer(i,j)
examinar los predecesores de los nodos i y j para identificar una corola B
crear un nuevo nodo b y definir $A_b^c = \cup_{k \in B} A_k^c$
etiquetar el nodo b como Par
añadir el nodo b a $\mathcal{L}$
actualizar $A_j^c \leftarrow A_j^c \cup \{b\}$ para cada $j \in A_b^c$
crear una lista con la información de los nodos de B
eliminar del grafo todos los nodos de B y los arcos incidentes en ellos

Algoritmo 11: Continuación de búsquedaNB.

Lema 5.3. *Si G contiene un camino aumentador desde en el nodo p al nodo q con respecto a un emparejamiento M, entonces G^c contiene un camino aumentador desde p (o el pseudonodo que lo contiene) hasta q con respecto al emparejamiento M^c.*

Estos dos lemas nos permiten concluir que el grafo contraído contiene un camino aumentador que empieza en el nodo p si, y solo si, el grafo original contiene también un camino aumentador que empiece en p. Como consecuencia, podemos asegurar que el algoritmo presentado para el problema de emparejamiento no bipartito de máxima cardinalidad encuentra un emparejamiento máximo.

Además, se puede probar que este algoritmo tiene un tiempo de ejecución, en el peor caso, de $O(n^3)$, pues cada proceso de búsquedaNB requiere un tiempo por ejecución de $O(n^2)$ y el algoritmo ejecuta este proceso un máximo de n veces.

Ejemplo de resolución

Ilustremos el funcionamiento del algoritmo presentado aplicándolo al siguiente ejemplo, donde partimos de un grafo G no bipartito y un emparejamiento $M = \{(3,5),(4,6),(7,8)\}$ de cardinalidad 3.

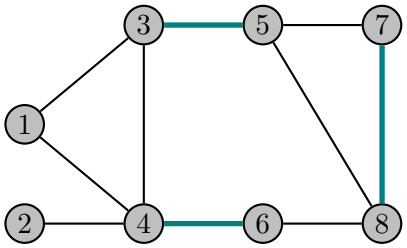
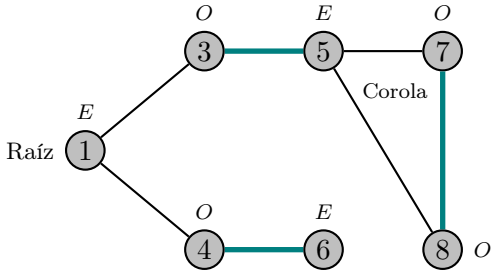
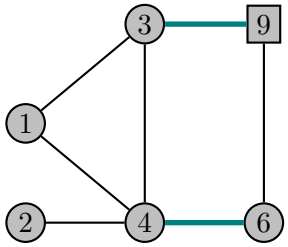


Figura 5.2: Problema de emparejamiento no bipartito.

Supongamos que el algoritmo selecciona el nodo 1 como nodo raíz. El algoritmo de búsqueda del camino aumentador examina los nodos en el siguiente orden: 1 (Impar), 3 (Par), 4 (Par), 5 (Impar), 6 (Impar), 7 (Par), 8 (Par). Examinando ahora el nodo 7, el algoritmo explora el arco $(7,8)$ y descubre la corola $5 \hookrightarrow 7 \hookrightarrow 8 \hookrightarrow 5$. Contraemos esta corola en el pseudo-nodo con el número 9 y obtenemos el siguiente grafo contraído.

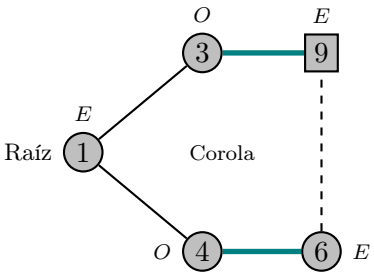


(a) Etiquetas asignadas.

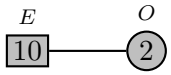


(b) Grafo contraído 1.

En este momento, el nodo 9 es el único nodo no examinado. Examinando la lista de adyacencia del nodo 9 descubrimos otra corola que abarca los nodos $1 \hookrightarrow 3 \hookrightarrow 9 \hookrightarrow 6 \hookrightarrow 4 \hookrightarrow 1$. Contraemos esta corola en el pseudo-nodo con el número 10 y obtenemos el siguiente grafo contraído.



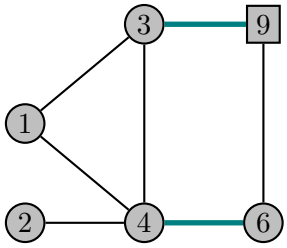
(c) Corola.



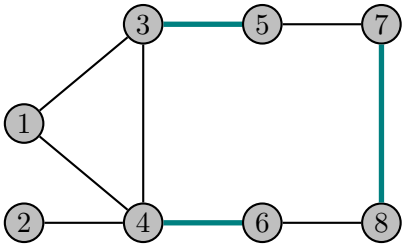
(d) Grafo contraído 2.

Ahora examinando el nodo 10, que es el pseudo-nodo que contiene al nodo raíz, se le asigna etiqueta Impar al nodo 2 que no está emparejado y por tanto descubrimos el camino aumentador $10 \hookrightarrow 2$.

Para identificar el camino aumentador en el grafo original expandimos los pseudo-nodos que formen parte del camino aumentador que encontramos. Para empezar expandimos el pseudo-nodo 10. El nodo 2 es adyacente al nodo 4 de la corola, por lo que al arco $(2, 4)$ le unimos el camino alternado par que va desde el nodo raíz 1 hasta el nodo 4, obteniendo así el camino $1 \hookrightarrow 3 \hookrightarrow 9 \hookrightarrow 6 \hookrightarrow 4 \hookrightarrow 2$. Por último expandimos el pseudo-nodo 9, obteniendo el camino aumentador en el grafo original $P = 1 \hookrightarrow 3 \hookrightarrow 5 \hookrightarrow 7 \hookrightarrow 8 \hookrightarrow 6 \hookrightarrow 4 \hookrightarrow 2$.



(e) Después de expandir el nodo 10.



(f) Después de expandir el nodo 9.

Actualizando el emparejamiento con la operación $M \leftarrow M \oplus P$ conseguimos el siguiente emparejamiento sobre el grafo original de cardinalidad 4, $M = \{(1, 3), (5, 7), (8, 6), (4, 2)\}$.

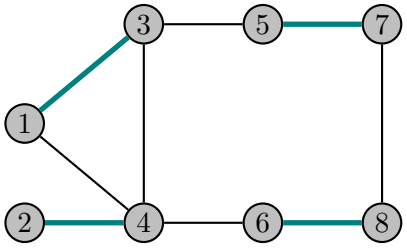


Figura 5.3: Problema de emparejamiento no bipartito.

5.2. El problema no bipartito ponderado

Sea $R = ((N, A), \boldsymbol{u}, \boldsymbol{c})$ una red con costes donde el grafo subyacente es no bipartito, no dirigido y que además cumple $u_{ij} = 1$ para todo $(i, j) \in A$. Un emparejamiento de coste mínimo es un emparejamiento sobre R con el menor coste posible. Hay varias variantes del problema no bipartito ponderado: *el problema de emparejamiento de mínimo coste y máxima cardinalidad* que busca un emparejamiento con el mayor número de arcos posibles y que, sujeto a esta restricción, tiene el menor coste posible; *el problema de emparejamiento de mínimo coste y cardinalidad k* (para un entero dado k); *el problema de emparejamiento de máximo coste*; etc. Cuando hablamos de problema de emparejamiento no bipartito ponderado nos referimos a cualquiera de estas versiones.

De este caso particular de problema de emparejamiento enunciaremos la existencia de un par de algoritmos eficientes para su resolución, pues su desarrollo en profundidad excedería la extensión adecuada para un trabajo de este tipo.

Se puede consultar en la Sección 11.3 de Papadimitriou and Steiglitz (1998) un estudio en profundidad de este problema de optimización en redes. En él se contempla el algoritmo primal-dual como una herramienta para transformar las versiones ponderadas de los problemas de emparejamiento en sus versiones de máxima cardinalidad. Encontramos pues un algoritmo que hace uso del primal-dual para resolver el problema no bipartito ponderado en un tiempo de $O(n^4)$.

Actualmente, los algoritmos más rápidos conocidos para resolver este problema son: un algoritmo de velocidad $O(nm + n^2 \log n)$ debido a Gabow (1990), y un algoritmo de velocidad $O(m \log(nC)\sqrt{n\alpha(m, n) \log n})$ debido a Gabow and Tarjan (1989).

Anexo A

Aplicaciones del problema de emparejamiento

Los problemas de emparejamiento surgen en una variedad muy amplia de contextos en la vida real. A continuación mostramos ejemplos de situaciones que podemos plantear, y posteriormente resolver, como problemas de emparejamiento de los que hablamos a lo largo del trabajo.

A.1. Recableando máquinas de escribir

Una compañía lleva años utilizando un tipo específico de máquinas de escribir eléctricas que perforan cintas de papel, con hasta 6 agujeros, para introducir datos en un ordenador. Así que puede hacer $2^6 = 64$ patrones binarios distintos teniendo en cuenta que, en cada una de esas 6 posiciones, puede perforar o no hacerlo. Las máquinas de escribir tienen 46 caracteres, cada uno de los cuales se corresponde con uno de los 64 patrones. Ahora bien, la compañía decide adquirir nuevos ordenadores que utilizan otro código distinto para representar los caracteres dentro del mismo modelo de papel perforado. Por ejemplo, si usamos 1 para representar un agujero y 0 para representar un no-agujero, la letra A se corresponde con 111100 en el código del ordenador antiguo y con 011010 en el código de los ordenadores nuevos. Dado que la máquina de escribir está adaptada al código del ordenador antiguo, el problema consiste en modificarla para que perfore el papel siguiendo el nuevo código.

Cada tecla de la máquina de escribir está conectada a una barra de acero, por lo que modificar el código que sigue esa tecla requiere cambios mecánicos en el sistema de barras de acero. Por ejemplo, fijándonos de nuevo en la letra A, deberíamos hacer tres cambios

correspondientes al primer, cuarto y quinto bit. Cuando se presiona una tecla, su barra activa seis travesaños (comunes a todas las teclas) que están conectados eléctricamente a seis perforadores. Intercambiar el cuarto y el quinto cable desde los travesaños a los perforadores equivaldría a intercambiar el cuarto y el quinto bit en el viejo código de todos los caracteres. Esto reduciría el número de cambios mecánicos necesarios para actualizar la letra A, pero seguramente aumentaría el número de cambios necesarios para alguna de las otras 45 teclas. Por tanto el problema que se plantea es el siguiente: ¿cómo conectar los cables desde los 6 travesaños a las 6 perforadoras de manera óptima? Es decir, buscamos una conexión que minimice el número de cambios mecánicos que se deben hacen en las barras de acero.

Esta situación se puede plantear como un **problema de asignación** como sigue: definimos la red $R = ((N_1 \cup N_2, A), \boldsymbol{u}, \boldsymbol{c})$, donde $N_1 = \{1, 2, 3, 4, 5, 6\}$ y $N_2 = \{1', 2', 3', 4', 5', 6'\}$, $A = N_1 \times N_2$ y $u_{ij'} = 1$ para todo $(i, j') \in A$. El coste del arco $(i, j') \in A$ es el número de teclas, de las 46 totales, en las cuales el bit i-ésimo del código antiguo difiere del bit j-ésimo en el código nuevo. Así, si asignamos el travesaño i a la perforadora j, $c_{ij'}$ es el número de cambios mecánicos que tendremos que hacer para grabar el bit i-ésimo de cada caracter correctamente. Así, el emparejamiento de menor coste entre los nodos de N_1 y N_2 minimizará el número de cambios mecánicos necesarios.

A.2. Asociando altavoces estéreo

Una compañía que manufactura altavoces estéreo debe asociar los altavoces individuales en parejas antes de poder venderlos como un pack. El funcionamiento de los altavoces depende de su respuesta de frecuencia. Para medir la calidad de los packs la compañía genera unos coeficientes de emparejamiento para cada par posible. Estos coeficientes se calculan sumando las diferencias entre las respuestas de los dos altavoces del pack a ciertas frecuencias. Las malas asociaciones producen un coeficiente alto, mientras que una asociación buena produce un coeficiente bajo.

La compañía sigue dos criterios diferentes para decidir las parejas de altavoces que pondrá a la venta: (1) encontrar tantos pares posibles cuyos coeficientes de emparejamiento no excedan un límite específico, o (2) emparejar altavoces dentro de unos límites específicos de manera que se minimice la suma total de sus coeficientes de emparejamiento. El primer criterio minimiza el número de pares que no cumplen las especificaciones, y por tanto el número de altavoces que tendrá que vender individualmente a un precio reducido. Este modelo es una aplicación del **problema de emparejamiento no bipartito de máxima cardinalidad** sobre un grafo no dirigido: los nodos representan los altavoces y los arcos

entre dos nodos representan que ese coeficiente de emparejamiento entá dentro de los límites especificados. El segundo modelo es una aplicación del **problema de emparejamiento no bipartito ponderado**.

A.3. Asignación de personal

En muchos contextos distintos deseamos asignar personas a objetos, trabajos, máquinas, etc. Cada asignación tiene un "valor" y queremos encontrar un emparejamiento que maximice la suma total de esos valores. A continuación veremos varios ejemplos relacionados con la asignación de personal, tanto bipartita como no bipartita.

1. Una empresa ha contratado n graduados para cubrir n plazas vacantes. Basándose en pruebas de aptitud, cualificaciones, etc. la empresa ha asignado a cada candidato i un índice de destreza u_{ij} para el trabajo j. El objetivo es identificar un emparejamiento que maximice la puntuación de destreza total de todos los trabajos. Este problema es claramente una aplicación del **problema de emparejamiento bipartito ponderado** o problema de asignación.

2. Un entrenador de natación debe seleccionar de sus ocho mejores nadadores a un equipo de cuatro para las competiciones de relevos, cada uno de los cuales nadará uno de los cuatro estilos (espalda, braza, mariposa y crol). El entrenador, conociendo los tiempos de todos los nadadores en cada estilo, debe identificar el equipo de los mejores cuatro nadadores. La suma de los tiempos obtenida al emparejar óptimamente los ocho nadadores a los cuatro estilos proporciona el mínimo tiempo factible para la carrera de relevos, y por tanto el equipo correspondiente es el mejor equipo. Nótese que en esta versión del **problema de asignación** se tiene $|N_1| > |N_2|$, sin embargo, añadiendo nodos artificiales que representarán a los nadadores que no compitan, se puede transformar fácilmente este problema en otro equivalente en el cual N_1 y N_2 tienen el mismo número de nodos.

3. Durante la Segunda Guerra Mundial, las Fuerzas Aéreas de Gran Bretaña contaban con muchos pilotos extranjeros que hablaban distintos idiomas y con diferentes niveles de entrenamiento. Tenían que asignar dos pilotos a cada avión de manera que se pudiesen entender y que tuviesen un nivel de entrenamiento similar. Además buscaban poder emparejar el mayor número de pilotos posibles. Para formular este problema como un **problema de emparejamiento no bipartito de máxima cardinalidad** definimos un grafo cuyos nodos representan a los pilotos, mientras que incluimos el arco (i, j) si los pilotos i y j son compatibles.

4. Supongamos que una aerolínea desea dividir sus $2p$ pilotos, ordenados por antigüedad, en m equipos cada uno de los cuales contendrá un capitán y un primer oficial. Además el capitán de cada equipo debe tener más antigüedad que el primer oficial. A cada piloto i se le asigna un indicador, α_i, de su efectividad como capitán y otro, β_i, que mide su efectividad como primer oficial. Se busca una asignación de pilotos que maximice la suma de los indicadores de efectividad de todos los grupos. Este problema es un claro ejemplo de **problema de emparejamiento no bipartito de máximo coste**, donde cada nodo representa un piloto y definimos el coste del arco (i, j) como $\alpha_j + \beta_i$ si el piloto j tiene más antigüedad que el i y como $\alpha_i + \beta_j$ en caso contrario.

A.4. El problema del camino más corto

En la Sección 12.7 de Ahuja et al. (1993) se describen algunas relaciones interesantes entre los emparejamientos y caminos. Se muestra cómo transformar el problema del camino más corto sobre redes dirigidas con capacidades arbitrarias en dos problemas de asignación. De esta forma se consigue el que es actualmente el mejor algoritmo para resolver este tipo de problemas. También se muestra cómo resolver el problema del camino más corto sobre redes no dirigidas con capacidades arbitrarias resolviendo un problema de emparejamiento no bipartito ponderado.

Bibliografía

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993): *Network flows: theory, algorithms and applications*, Prentice Hall.

Berge, C. (1957): "Two theorems in graph theory," *Princeton University*.

Gabow, H. N. (1990): "Data structures for weighted matching and nearest common ancestors with linking," in *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics.

Gabow, H. N. and R. E. Tarjan (1989): "Faster scaling algorithms for general graph matching problems," Tech. rep., PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE.

González-Díaz, J. (2018): *Programación lineal y entera*, USC.

Papadimitriou, C. H. and K. Steiglitz (1998): *Combinatorial optimization: algorithms and complexity*, Dover.